	L #	Hits	Search Text	DBs
1	L4	493	(select$3 execut$3) near20 (thread context) near20 (sequential$3 robin)	USPAT; US-PGPUB
2	L5	1692	(instruction adj2 (pointer counter) ip (address near10 (fetch$3 prefetch$3))) near20 (thread context)	USPAT; US-PGPUB
3	L6	75	4 and 5	USPAT; US-PGPUB
4	L9	1	7 and 8	EPO; JPO; DERWENT; IBM_TDB
5	L10	1889	(instruction adj2 (pointer counter) ip ((instruction address) near10 (fetch$3 prefetch$3))) near20 (thread context)	USPAT; US-PGPUB
6	L11	19	4 and 10 not 6	USPAT; US-PGPUB
7	L12	114	(instruction adj2 (pointer counter) ip ((instruction address) near10 (fetch$3 prefetch$3))) near20 (thread context)	EPO; JPO; DERWENT; IBM_TDB
8	L13	2	7 and "13"	EPO; JPO; DERWENT; IBM_TDB
9	L14	3	7 and 12	EPO; JPO; DERWENT; IBM_TDB
10	L8	89	(instruction adj2 (pointer counter) ip (address near10 (fetch$3 prefetch$3))) near20 (thread context)	EPO; JPO; DERWENT; IBM_TDB
11	L7	34	(select$3 execut$3) near20 (thread context) near20 (sequential$3 robin)	EPO; JPO; DERWENT; IBM_TDB
12	L20	45	(plural plurality multiple multiplicity several number) adj2 (instruction adj2 (pointer counter) ip ((instruction address) near10 (fetch$3 prefetch$3))) near20 (multithread$3 thread context) not (6 11)	USPAT; US-PGPUB
13	L21	1	(plural plurality multiple multiplicity several number) adj2 (instruction adj2 (pointer counter) ip ((instruction address) near10 (fetch$3 prefetch$3))) near20 (multithread$3 thread context) not 14	EPO; JPO; DERWENT; IBM_TDB

(PRIOR ART)
FIG. 3B

opf[8:4] \ opf[3:0]	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	—	FMOVs	FMOVd	FMOVq	—	FNEGs	FNEGd	FNEGq	—	FABSs	FABSd	FABSq	—	—	—	—
01	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
02	—	—	—	—	—	—	—	—	—	FSQRTs	FSQRTd	FSQRTq	—	—	—	—
03	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
04	—	FADDs	FADDd	FADDq	—	FSUBs	FSUBd	FSUBq	—	FMULs	FMULd	FMULq	—	FDIVs	FDIVd	FDIVq
05	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
06	—	—	—	—	—	—	—	—	—	FsMULd	—	—	—	—	FdMULq	—
07	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
08	—	FsTOx	FdTOx	FqTOx	FxTOs	—	—	—	FxTOd	—	—	—	FxTOq	—	—	—
09	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
0A	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
0B	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
0C	—	—	—	—	FiTOs	—	FdTOs	FqTOs	FiTOd	FsTOd	—	FqTOd	FiTOq	FsTOq	FdTOq	—
0D	—	FsTOi	FdTOi	FqTOi	—	—	—	—	—	—	—	—	—	—	—	—
0E..1F	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

FIG. 4

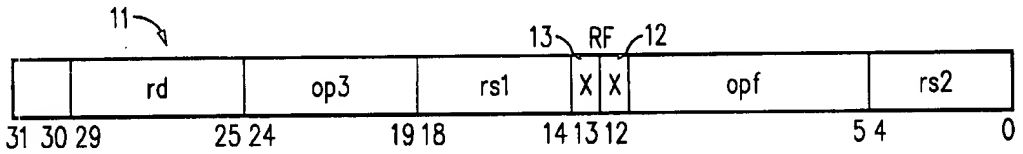
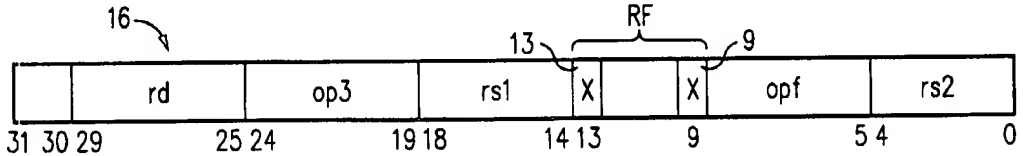


FIG. 5



	Document ID	U	Title	Current OR
1	US 20040054990 A1	☐	Post-pass binary adaptation for software-based speculative precomputation	717/124
2	US 20040034759 A1	☒	Multi-threaded pipeline with context issue rules	712/1
3	US 20040032865 A1	☒	Apparatus and method for establishing a call connection state in a packet data communication system	370/367
4	US 20030212660 A1	☒	Database scattering system	707/1
5	US 20030189565 A1	☒	Single semiconductor graphics platform system and method with skinning, swizzling and masking capabilities	345/418
6	US 20030112956 A1	☒	Transferring a call to a backup according to call context	379/221.01
7	US 20030112246 A1	☒	Blending system and method in an integrated computer graphics pipeline	345/519
8	US 20030112245 A1	☒	Single semiconductor graphics platform	345/506
9	US 20030103054 A1	☒	Integrated graphics processing unit with antialiasing	345/506
10	US 20030103050 A1	☒	Masking system and method for a graphics processing framework embodied on a single semiconductor platform	345/426
11	US 20030097548 A1	☒	Context execution in pipelined computer processor	712/228
12	US 20030061601 A1	☒	Data processing apparatus and method, computer program, information storage medium, parallel operation apparatus, and data processing system	717/144
13	US 20030041110 A1	☒	System, Method and Structure for generating and using a compressed digital certificate	709/206
14	US 20030038808 A1	☒	Method, apparatus and article of manufacture for a sequencer in a transform/lighting module capable of processing multiple independent execution threads	345/506
15	US 20030037228 A1	☒	System and method for instruction level multithreading scheduling in a embedded processor	712/245
16	US 20030034975 A1	☒	Lighting system and method for a graphics processor	345/426
17	US 20030020720 A1	☒	Method, apparatus and article of manufacture for a sequencer in a transform/lighting module capable of processing multiple independent execution threads	345/506

invention. Each bundle includes a plurality of instructions and a template field grouped together in a N-bit field. The instructions are located in instruction slots of the N-bit field, with the template field specifying a mapping of the instruction slots to the execution unit types. Other encodings or grouping of instructions are also possible, and are considered within the scope of the present invention.

With continuing reference to the embodiment of FIG. 1, the input to rotator buffer **24** can be zero, one, or two instruction bundles. The number of bundles issued to the EXP stage that are consumed by the execution core may also be zero, one, or two bundles. Note, however, that information about number of bundles available in the number of bundles consumed by the execution core is not available to the rotate pipeline stage logic until the middle of the CPU clock cycle.

Rotate buffer **24** is organized as a circular, wrap-around buffer, which is implemented in the embodiment of FIG. **1** with eight entries, labeled IB_0–IB_7. Rotate buffer **24** has an associated starting write pointer **23** that points to a starting entry for filling (i.e., writing) of instruction bundles into the buffer. Fill operations are controlled by fill control logic block **22**. Filling operations take place in-order; that is, according to the original code sequence of the program. After each entry location of buffer **24** gets written with an instruction bundle, fill pointer **23** advances in one position in the buffer stack.

On the readout side, a starting read pointer (i.e., drain pointer) **25** is also associated with buffer **24**. FIG. **1** shows the starting read pointer point to IB_0, with up to four instruction bundles (IB_0–IB_3) being read out each clock cycle. The speculative read of the buffer is performed by reading out four consecutive bundles starting from the starting read pointer location, and sending the instruction data to the alignment multiplexers **26** and **27**. Depending on the number of bundles consumed by the execution core, the number of bundles available from buffer **24**, and the number read from cache **11**, the appropriate bundles from the speculative read and the bypass case are selected from each alignment multiplexer.

According to the embodiment of FIG. **1**, the number of bundles consumed by the execution core is at most two; the number of bundles read from buffer **24** is at most four; and the number read from cache **11** is at most two. Drain control logic unit **28** receives information about the number of bundles consumed in the execution core. Once the bundle consumption information becomes available, two different bypass combinations and four consecutive bundles from rotator buffer **24** are speculatively read out and then selected.

Alignment multiplexers **26** and **27** are an in-order output to the dispersal stage to maintain the FIFO scheme. The alignment multiplexers perform any needed rotation, selecting the appropriate bundles to be latched in dispersal latches **31** & **32**, respectively. Each of the alignment multiplexers has six inputs, two of which are the instruction cache way multiplexer (21) outputs, with each bypass instance consisting of two instruction bundles. Also included among the inputs to the alignment multiplexers are the four other consecutive instruction bundles read out from rotator buffer **24**.

As the execution engine of the processor consumes instruction bundles, next in-order bundles get consecutively latched into the dispersal latches through the alignment multiplexers. In operation, drain pointer **25** basically trails fill pointer **23**, wrapping around the end of the rotate buffer **24**.

With reference to FIG. **2**, the bus wire routing of the single read port of rotate buffer **24** is shown. Due to the speculative read, the wiring bundles, and their associated tagging information, an ordinary practitioner might expect that a large routing area would be required to implement the logic of the rotate buffer pipestage. One of the important aspects of the present invention is that rotate buffer **24** utilizes a shared read port design that optimizes usage of silicon area. In other words, the single read port buffer **24** is used to read out four consecutive instruction bundles to alignment multiplexers **26** and **27**; however, the number of read ports of the register file memory cells is reduced from two ports per cell to one per cell. This reduces the transistor count from twenty transistors per cell to fourteen transistors per memory cell, and also reduces the number of read enables that need to be routed to each cell from four to one. Altogether, for the described embodiment, there are eight buffers per bit, with a total of 2032 memory cells and eight (rather than thirty-two) read enables.

As can be seen in FIG. **2**, the 8-entry register file of buffer **24** is organized such that every fourth entry shares the single read port bus. For example, IB_0 shares the bus with IB_4, IB_1 shares the bus with IB_5, IB_2 shares the bus with IB_6, and IB_3 shares the bus with IB_7. This approach optimizes area savings of by taking advantage of the consecutive entries read from the buffer.

Instead of implementing buffer **24** as a register file, it could also be implemented as a memory array, with an associated address index. Use further appreciated that the concept of the present invention can be extended to include a single write port, rather than the dual write port scheme shown in FIG. **1**. In the case of a single write port a simple on/even scheme may be employed in which odd entries (IB_1, IB_3, IB_5, and IB_7) are organized in one group, and even entries (IB_0, IB_2, IB_4, and IB_6) are organized in another group.

It should be understood that although the present invention has been described in conjunction with specific embodiments, numerous modifications and alterations could be made without departing from the scope of the present invention. Accordingly, the specification and drawings are to be regarded in an illustrative rather than a restrictive sense.

I claim:

1. A processor comprising:

an instruction cache that stores a cache line of instructions;

an execution engine for executing the instructions;

a circular buffer having a plurality of entries;

a first logic circuit that divides the cache line into instruction bundles, each of which is written into an entry of the circular buffer; and

a second logic circuit that reads out a number of consecutive instruction bundles from the circular buffer for dispersal to the execution engine.

2. A processor comprising:

an instruction cache that stores a cache line of instructions;

an execution engine for executing the instructions;

a buffer having a starting write pointer and a starting read pointer;

a first logic circuit that divides the cache line into instruction bundles, each of which is written into an entry of the buffer; and

a second logic circuit that reads out a number of consecutive instruction bundles from the buffer for dispersal to the execution engine.

	Document ID	U	Title	Current OR
18	US 20030009694 A1	☒	Hardware architecture, operating system and network transport neutral system, method and computer program product for secure communications and messaging	713/201
19	US 20030009648 A1	☒	Apparatus for supporting a logically partitioned computer system	711/202
20	US 20030005262 A1	☒	Mechanism for providing high instruction fetch bandwidth in a multi-threaded processor	712/207
21	US 20020199096 A1	☒	System and method for secure unidirectional messaging	713/153
22	US 20020199001 A1	☒	System and method for conducting a secure response communication session	709/227
23	US 20020196935 A1	☒	Common security protocol structure and mechanism and system and method for using	380/37
24	US 20020196259 A1	☒	Single semiconductor graphics platform with blending and fog capabilities	345/506
25	US 20020194501 A1	☒	System and method for conducting a secure interactive communication session	713/201
26	US 20020194483 A1	☒	System and method for authorization of access to a resource	713/185
27	US 20020180740 A1	☒	Clipping system and method for a single graphics semiconductor platform	345/506
28	US 20020178360 A1	☒	System and method for communicating a secure unidirectional response message	713/170
29	US 20020165912 A1	☒	Secure certificate and system and method for issuing and using same	709/203
30	US 20020108003 A1	☒	COMMAND QUEUEING ENGINE	710/39
31	US 20020105519 A1	☒	Clipping system and method for a graphics processing framework embodied on a single semiconductor platform	345/426
32	US 20020047846 A1	☒	System, method and computer program product for performing a scissor operation in a graphics processing framework embodied on a single semiconductor platform	345/522
33	US 20020038416 A1	☒	System and method for reading and writing a thread state in a multithreaded central processing unit	712/228
34	US 20020027553 A1	☒	Diffuse-coloring system and method for a graphics processing framework embodied on a single semiconductor platform	345/426

3. The processor of claim 1 wherein the number of consecutive instruction bundles is equal to 4.

4. The processor of claim 1 further comprising an instruction bypass path coupled between the first and second logic circuits.

5. The processor of claim 1 wherein the plurality of entries is equal to 8 and the cache line comprises 32 bytes.

6. A processor comprising:

an instruction cache that stores a cache line of instructions;

an execution core for executing the instructions;

a first multiplexer circuit that divides the cache line into instruction bundles;

a N-entry buffer having a plurality of entries and a single read port, the buffer being organized such that every ith entry shares the single read port, the buffer including a fill pointer that points to a starting entry for filling with an instruction bundle provided by the first logic circuit; and

a second multiplexer circuit that aligns and presents each instruction bundle to the execution core;

drain control logic that selects an appropriate number of instruction bundles read out from the buffer.

7. The processor of claim 6 wherein the cache line comprises 32 bytes.

8. The processor of claim 7 wherein N=8 and i=4.

9. The processor of claim 6 wherein the number of consecutive instructions is equal to 4.

10. The processor of claim 6 further comprising an instruction bypass path coupled between the first and second multiplexer circuits.

11. The processor of claim 10 wherein the instruction bypass path transfers a pair of instruction bundles.

12. The processor of claim 6 wherein the second multiplexer circuit includes latches that latch the instruction bundles prior to dispersal to the execution core.

13. The processor of claim 10 wherein the instruction bypass path transfers a pair of instruction bundles.

14. The processor of claim 10 wherein the buffer comprises a circular buffer.

15. A logic circuit which supplies instructions from an instruction cache to the execution core of a processor comprising:

a circular buffer having N-entries and a read pointer and a write pointer, the circular buffer having a pair of write ports and a single read port, the circular buffer being organized such that every ith entry shares the single read port;

way multiplexer logic for dividing a cache line of the instruction cache into first and second instruction bundles, the first and second instruction bundles being written into consecutive entries of the circular buffer starting from the write pointer;

alignment multiplexer logic for aligning and presenting individual ones of the instruction bundles to the execution core of the processor, wherein j consecutive instruction bundles are read out of the circular buffer at a time starting from the read pointer; and

control logic for selecting instruction bundles to be output from the alignment multiplexer logic.

16. The logic circuit of claim 15 wherein the alignment multiplexer logic comprises:

first and second latches; and

first and second multiplexers coupled to the single read port of the circular buffer, the first and second multiplexers each outputting an instruction bundle which is respectively latched in the first and second latches prior to dispersal to the execution core.

17. The logic circuit of claim 16 further comprising:

an instruction bypass path coupled between the way multiplexer logic and the alignment multiplexer logic; and

wherein the control logic selects appropriate instruction bundles to be output from the first and second multiplexers depending on a number of instruction bundles consumed by the execution core, and a number of instruction bundles available from the circular buffer.

18. The logic circuit of claim 17 wherein the number of instruction bundles consumed by the execution core in a clock cycle is at most 2.

19. The logic circuit of claim 17 wherein the number of instruction bundles available from the circular buffer in a clock cycle is at most 4.

* * * * *

	Document ID	U	Title	Current OR
35	US 20020013866 A1	☒	RETRIEVAL CHANNELS FOR A DATA CONTROLLER	710/22
36	US 20020002667 A1	☒	System and method for instruction level multithreading in an embedded processor using zero-time context switching	712/228
37	US 20010018701 A1	☒	Performance enhancements for threaded servers	718/105
38	US 20010017626 A1	☒	Graphics processing unit with transform module capable of handling scalars and vectors	345/501
39	US 20010005209 A1	☒	Method, apparatus and article of manufacture for a transform module in a graphics processor	345/506
40	US 6687812 B1	☒	Parallel processing apparatus	712/230
41	US 6650331 B2	☒	System, method and computer program product for performing a scissor operation in a graphics processing framework embodied on a single semiconductor platform	345/522
42	US 6650330 B2	☒	Graphics system and method for processing multiple independent execution threads	345/506
43	US 6650325 B1	☒	Method, apparatus and article of manufacture for boustrophedonic rasterization	345/426
44	US 6597356 B1	☒	Integrated tessellator in a graphics processing unit	345/423
45	US 6577309 B2	☒	System and method for a graphics processing framework embodied utilizing a single semiconductor platform	345/426
46	US 6573900 B1	☒	Method, apparatus and article of manufacture for a sequencer in a transform/lighting module capable of processing multiple independent execution threads	345/537
47	US 6564267 B1	☒	Network adapter with large frame transfer emulation	709/250
48	US 6515671 B1	☒	Method, apparatus and article of manufacture for a vertex attribute buffer in a graphics processor	345/506
49	US 6504542 B1	☒	Method, apparatus and article of manufacture for area rasterization using sense points	345/441
50	US 6470443 B1	☒	Pipelined multi-thread processor selecting thread instruction in inter-stage buffer based on count information	712/205
51	US 6462737 B2	☒	Clipping system and method for a graphics processing framework embodied on a single semiconductor platform	345/426
52	US 6452595 B1	☒	Integrated graphics processing unit with antialiasing	345/426
53	US 6449666 B2	☒	One retrieval channel in a data controller having staging registers and a next pointer register and programming a context of a direct memory access block	710/23
54	US 6438671 B1	☒	Generating partition corresponding real address in partitioned mode supporting system	711/173
55	US 6427161 B1	☒	Thread scheduling techniques for multithreaded servers	718/102

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,539,469 B1
DATED : March 25, 2003
INVENTOR(S) : Jesse Pan

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 1,
Line 30, delete "modem", insert -- modern --.

Signed and Sealed this

Second Day of September, 2003

JAMES E. ROGAN
Director of the United States Patent and Trademark Office

	Document ID	U	Title	Current OR
56	US 6417851 B1	☒	Method and apparatus for lighting module in a graphics processor	345/426
57	US 6353439 B1	☒	System, method and computer program product for a blending operation in a transform module of a computer graphics pipeline	345/561
58	US 6342888 B1	☒	Graphics processing unit with an integrated fog and blending operation	345/426
59	US 6336150 B1	☒	Apparatus and method for enhancing data transfer rates using transfer control blocks	710/5
60	US 6324594 B1	☒	System for transferring data having a generator for generating a plurality of transfer extend entries in response to a plurality of commands received	710/5
61	US 6223202 B1	☒	Virtual machine pooling	718/102
62	US 6212542 B1	☒	Method and system for executing a program within a multiscalar processor by processing linked thread descriptors	718/102
63	US 6198488 B1	☒	Transform, lighting and rasterization system embodied on a single semiconductor platform	345/426
64	US 6073159 A	☒	Thread properties attribute vector based thread selection in multithreading processor	718/103
65	US 5961639 A	☒	Processor and method for dynamically inserting auxiliary instructions within an instruction stream during execution	712/242
66	US 5953520 A	☒	Address translation buffer for data processing system emulation mode	703/26
67	US 5949494 A	☒	Dedicated context-cycling computer with timed context	712/228
68	US 5913925 A	☒	Method and system for constructing a program including out-of-order threads and processor and method for executing threads out-of-order	712/206
69	US 5887166 A	☒	Method and system for constructing a program including a navigation instruction	718/102
70	US 5812811 A	☒	Executing speculative parallel instructions threads with forking and inter-thread communication	712/216
71	US 5781776 A	☒	Industrial controller permitting program editing during program execution	717/130
72	US 5742822 A	☒	Multithreaded processor which dynamically discriminates a parallel execution and a sequential execution of threads	718/102
73	US 5724565 A	☒	Method and system for processing first and second sets of instructions by first and second types of processing systems	712/245
74	US 5357617 A	☒	Method and apparatus for substantially concurrent multiple instruction thread processing by a single pipeline processor	712/245
75	US 4442484 A	☒	Microprocessor memory management and protection mechanism	711/163

HPS Trailer Page
for

Walk-Up_Printing

UserID: f

Printer: cpk2_2c21_gbkbptr

Summary

Document	Pages	Printed	Missed	Copies
US006292845	10	10	0	1
US006324639	53	53	0	1
US006539469	7	7	0	1
US005978896	9	9	0	1
US006065110	9	9	0	1
US20010052053	55	55	0	1
US006691221	19	19	0	1
Total (7)	162	162	0	-

	Document ID	U	Title	Current OR
1	US 20040 01568 4 A1	☐	Method, apparatus and computer program product for scheduling multiple threads for a processor	712/245
2	US 20030 23352 1 A1	☒	Method and apparatus for converting memory instructions to prefetch operations during a thread switch window	711/137
3	US 20030 23339 4 A1	☒	Method and apparatus for ensuring fairness and forward progress when executing multiple threads of execution	718/107
4	US 20030 04651 7 A1	☒	Apparatus to facilitate multithreading in a computer processor pipeline	712/214
5	US 20030 02875 5 A1	☒	Interprocessor register succession method and device therefor	712/216
6	US 20030 01868 4 A1	☒	Multi-thread execution method and parallel processor system	718/102
7	US 20030 01447 2 A1	☒	Thread ending method and device and parallel processor system	718/107
8	US 20030 01447 1 A1	☒	Multi-thread execution method and parallel processor system	718/107
9	US 20010 05645 6 A1	☒	PRIORITY BASED SIMULTANEOUS MULTI-THREADING	718/103
10	US 66584 47 B2	☒	Priority based simultaneous multi-threading	718/103
11	US 63634 53 B1	☒	Parallel processor with redundancy of processor pairs	711/2
12	US 62405 08 B1	☒	Decode and execution synchronized pipeline processing using decode generated memory read queue with stop entry to allow execution generated memory read	712/219
13	US 61611 66 A	☒	Instruction cache for multithreaded processor	711/125
14	US 58729 85 A	☒	Switching multi-context processor and method overcoming pipeline vacancies	710/1
15	US 58225 78 A	☒	System for inserting instructions into processor instruction stream in order to perform interrupt processing	712/244
16	US 55533 05 A	☒	System for synchronizing execution by a processing element of threads within a process using a state indicator	718/106
17	US 55420 58 A	☒	Pipelined computer with operand context queue to simplify context-dependent execution flow	713/502
18	US 55353 61 A	☒	Cache block replacement scheme based on directory control bit set/reset and hit/miss basis in a multiheading multiprocessor environment	711/145
19	US 54044 69 A	☒	Multi-threaded microprocessor architecture utilizing static interleaving	712/215

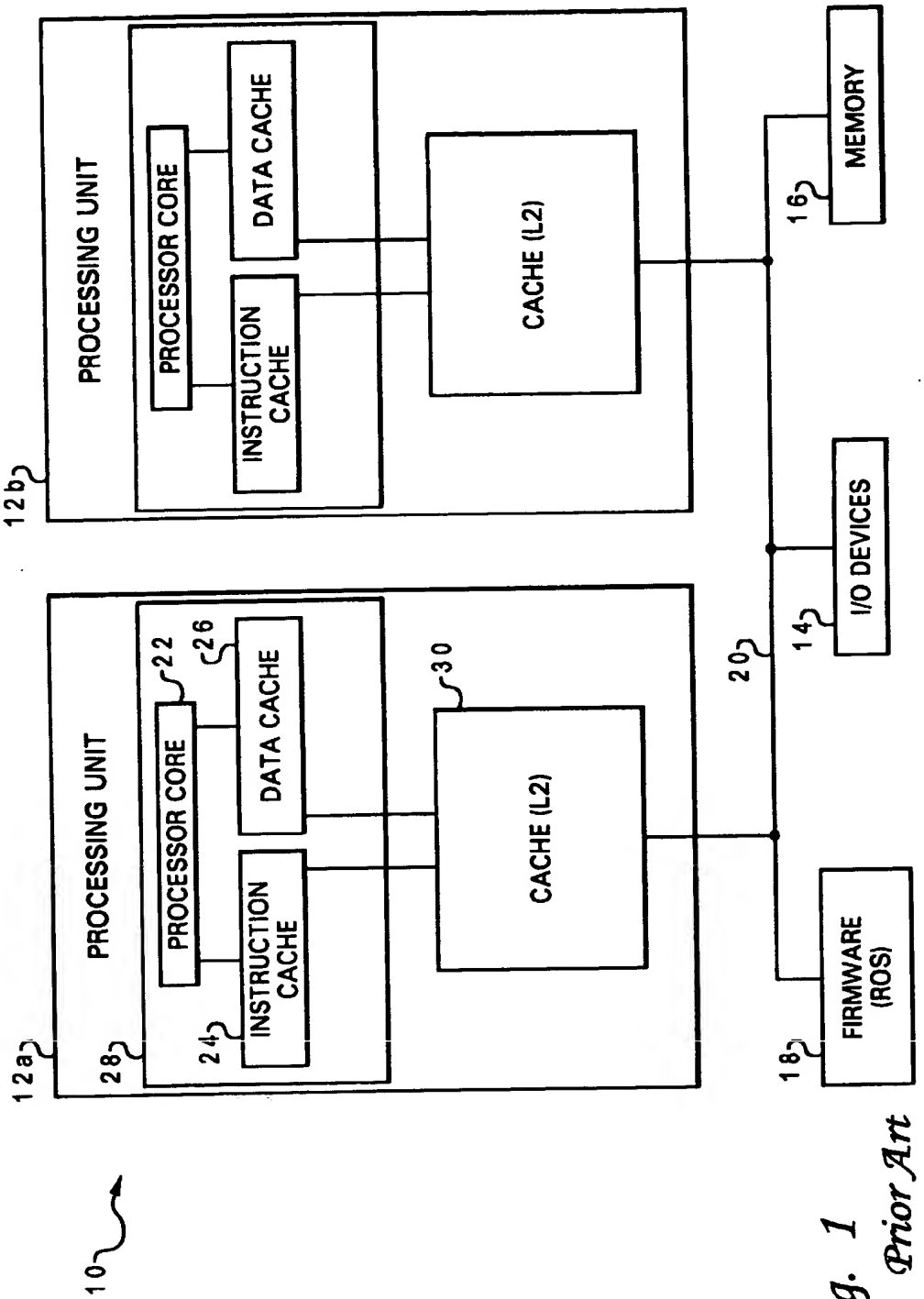


Fig. 1 Prior Art

	Document ID	U	Title	Current OR
1	DE 10110 504 A1	☐	Use of threads in connection with processor and accessible data by transferring execution control to next thread in queue if first thread is blocked	
2	US 6625635 B	☐	Threads scheduling method for multi-threaded data processing system, involves allocating prespecified number of instructions to instruction counter which counts execution of instruction by threads sequentially	
3	DE 10110 504 A	☐	Use of threads in connection with processor and accessible data by transferring execution control to next thread in queue if first thread is blocked	

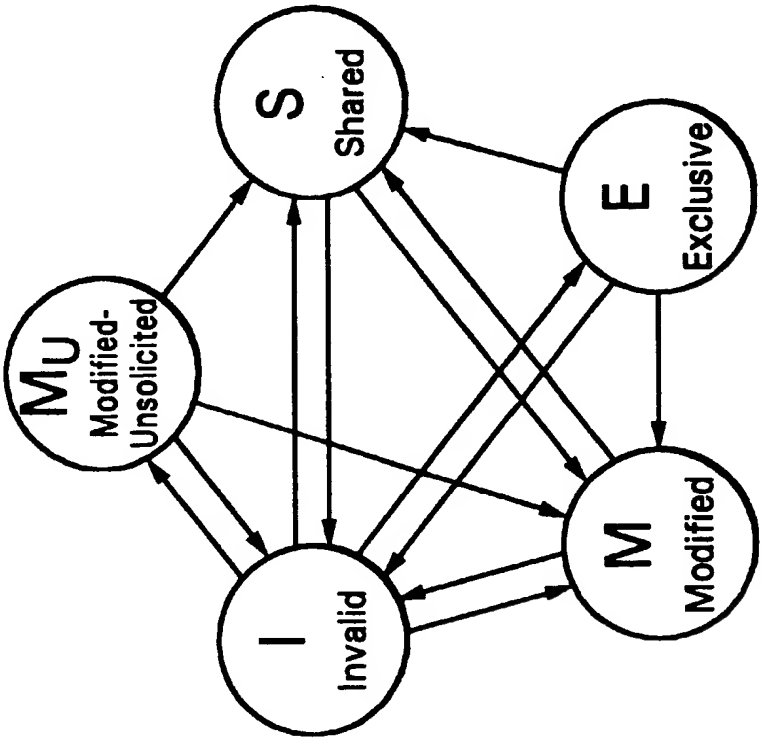


Fig. 3

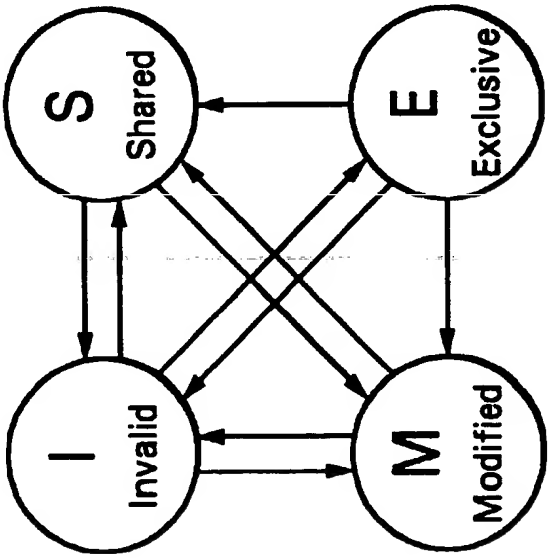


Fig. 2
Prior Art

	L #	Hits	Search Text	DBs
1	L4	493	(select$3 execut$3) near20 (thread context) near20 (sequential$3 robin)	USPAT; US-PGPUB
2	L5	1692	(instruction adj2 (pointer counter) ip (address near10 (fetch$3 prefetch$3))) near20 (thread context)	USPAT; US-PGPUB
3	L6	75	4 and 5	USPAT; US-PGPUB
4	L7	34	(select$3 execut$3) near20 (thread context) near20 (sequential$3 robin)	EPO; JPO; DERWENT; IBM_TDB
5	L8	89	(instruction adj2 (pointer counter) ip (address near10 (fetch$3 prefetch$3))) near20 (thread context)	EPO; JPO; DERWENT; IBM_TDB
6	L9	1	7 and 8	EPO; JPO; DERWENT; IBM_TDB
7	L10	1889	(instruction adj2 (pointer counter) ip ((instruction address) near10 (fetch$3 prefetch$3))) near20 (thread context)	USPAT; US-PGPUB
8	L11	19	4 and 10 not 6	USPAT; US-PGPUB
9	L12	114	(instruction adj2 (pointer counter) ip ((instruction address) near10 (fetch$3 prefetch$3))) near20 (thread context)	EPO; JPO; DERWENT; IBM_TDB
10	L13	2	7 and "13"	EPO; JPO; DERWENT; IBM_TDB
11	L14	3	7 and 12	EPO; JPO; DERWENT; IBM_TDB

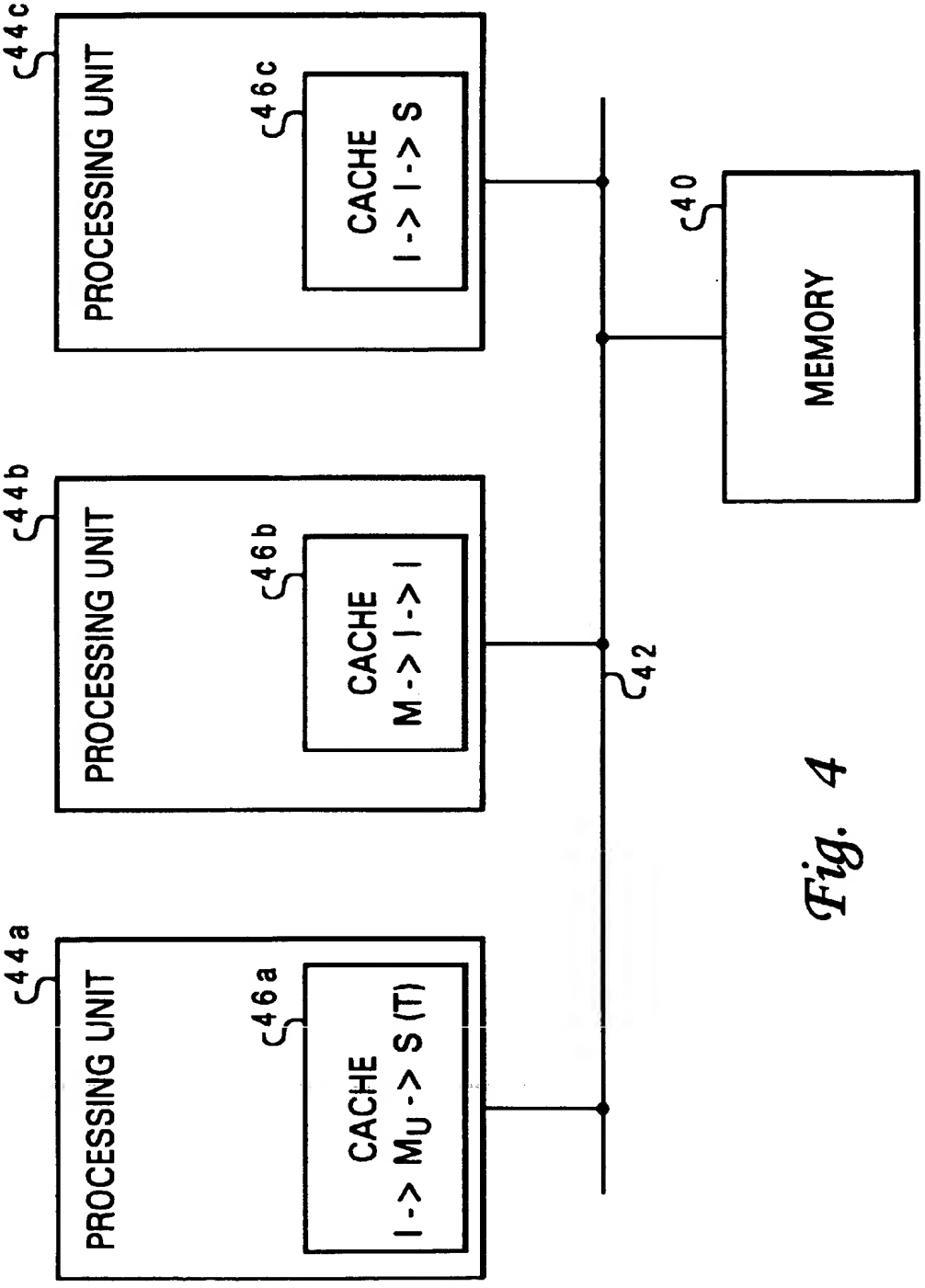

44c
PROCESSING UNIT
46c
CACHE
I -> I -> S
40
MEMORY
44b
PROCESSING UNIT
46b
CACHE
M -> I -> I
42
44a
PROCESSING UNIT
46a
CACHE
I -> M_U -> S (T)

Fig. 4

	Document ID	U	Title	Current OR
1	JP 2003259440 A	☐	MOBILE PORTABLE TERMINAL AND MANAGEMENT METHOD FOR PRESERVED PDP (PACKET DATA PROTOCOL) CONTEXT USED THEREFOR	
2	JP 2003174671 A	☒	MOBILE TERMINAL AND METHOD FOR MANAGING PDP CONTEXT IN ON-STATE AT ALL TIMES	
3	JP 2002141931 A	☒	ROUTER AND ROUTE CONTROL METHOD	
4	JP 2002009938 A	☒	SYSTEM AND METHOD FOR MULTISERVICE COMMUNICATION	
5	JP 2002141931 A	☒	ROUTER AND ROUTE CONTROL METHOD	
6	JP 2002009938 A	☒	SYSTEM AND METHOD FOR MULTISERVICE COMMUNICATION	
7	JP 06168117 A	☒	DATA PROCESSOR	
8	JP 04288632 A	☒	ONE-CHIP MICROCOMPUTER	
9	JP 04097434 A	☒	CONTROL SYSTEM FOR PROCESSING ASYNCHRONOUS EVENT	
10	JP 03186572 A	☒	THREAD HANDLING METHOD FOR TAIL END	
11	JP 03141435 A	☒	PROCESS SWITCHING SYSTEM	
12	JP 406206309 A	☒	TWO-PIECE CLAMPING DEVICE	
13	JP 01292430 A	☒	PARALLEL PROCESSING PROCESSOR	
14	WO 3107616 A1	☒	METHOD AND APPARATUS FOR INTERNET PROTOCOL HEADERS COMPRESSION INITIALIZATION	
15	WO 3092314 A1	☒	OPTIMIZED INFORMATION TRANSFER ASSOCIATED WITH RELOCATION OF AN IP SESSION IN A MOBILE COMMUNICATIONS SYSTEM	
16	WO 3055170 A1	☒	METHOD AND SYSTEM FOR SECURE HANDLING OF ELECTRONIC BUSINESS TRANSACTIONS ON THE INTERNET	
17	EP 1322090 A2	☒	Mechanism for simplifying roaming in a communications system	
18	WO 2087189 A1	☒	SYSTEM FOR EMULATING A TERMINAL WHICH IS ADAPTED TO ACCESS A TELECOMMUNICATION NETWORK FROM A TERMINAL WHICH IS INADAPTED TO SAID ACCESS	
19	WO 2054811 A1	☒	POSITIONING OF TERMINAL EQUIPMENT	
20	EP 1213892 A2	☒	System and method for implementing a client side HTTP stack	

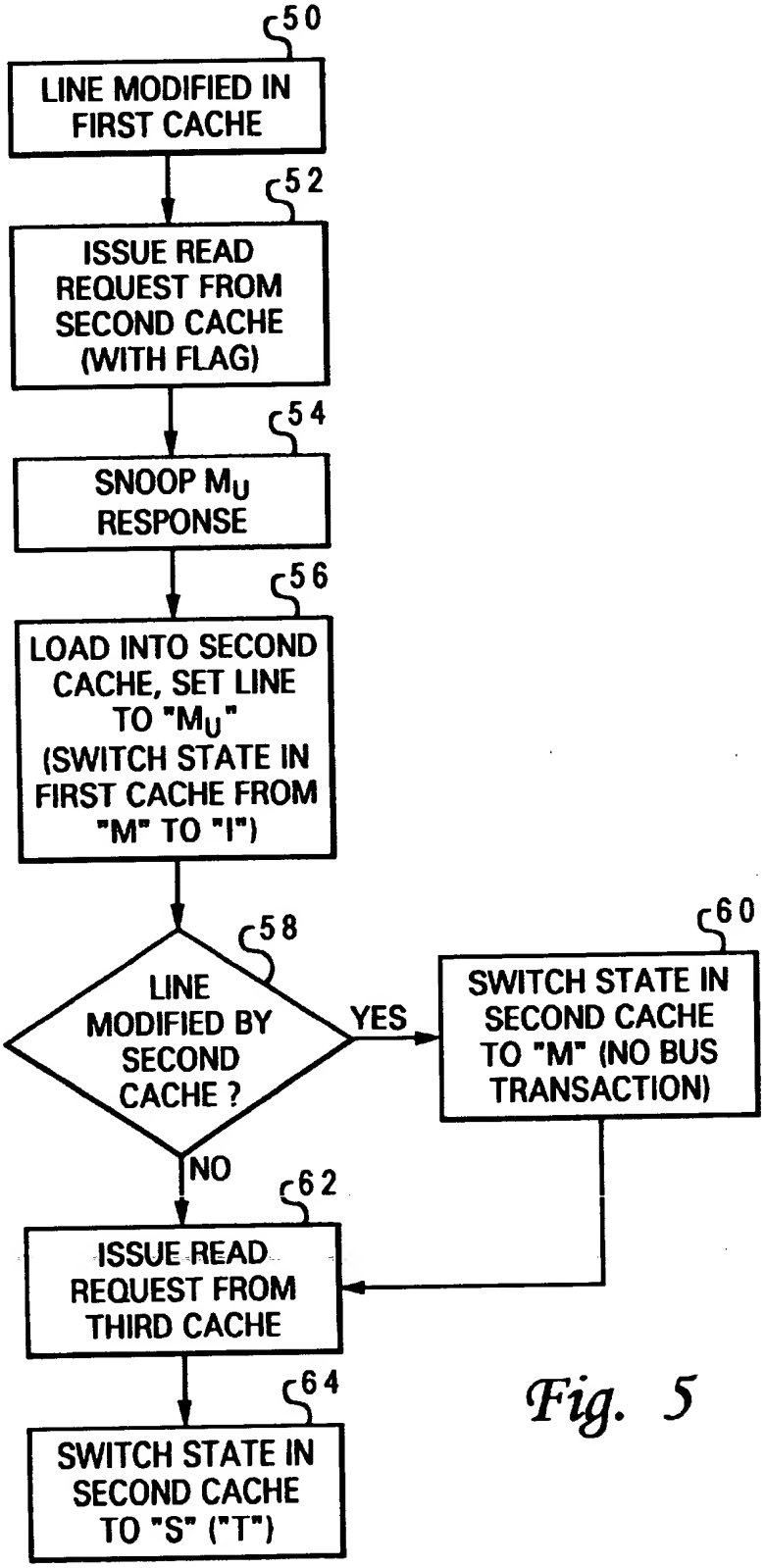

50
LINE MODIFIED IN FIRST CACHE
52
ISSUE READ REQUEST FROM SECOND CACHE (WITH FLAG)
54
SNOOP M_U RESPONSE
56
LOAD INTO SECOND CACHE, SET LINE TO "M_U" (SWITCH STATE IN FIRST CACHE FROM "M" TO "I")
58
LINE MODIFIED BY SECOND CACHE ?
YES
60
SWITCH STATE IN SECOND CACHE TO "M" (NO BUS TRANSACTION)
NO
62
ISSUE READ REQUEST FROM THIRD CACHE
64
SWITCH STATE IN SECOND CACHE TO "S" ("T")

Fig. 5

	Document ID	U	Title	Current OR
21	EP 1133140 A2	☒	Multi-service communication system and method	
22	EP 942365 A2	☒	Context controller having instruction-based time slice task switching capability and processor employing the same	
23	WO 9737303 A1	☒	TITLE DATA NOT AVAILABLE	
24	EP 580109 A2	☒	Data acces in a RISC digital signal processor.	
25	EP 134386 A2	☒	Method and apparatus for executing object code instructions compiled from a high-level language source.	
26	NNRD429148	☒	Controlled Server Side Execution Environment	
27	NNRD42676	☒	Multiple Terminal Simulation Tool Utilizing Multiple Separate IP Addresses	
28	NN950541	☒	Memory Refresh Schemes For Transmission Control Protocol/Internet Protocol	
29	NN940425	☒	Compound Document Client/Server Message Part Handler	
30	NN949521	☒	Fragment Recovery When Implementing a Datagram Fragmentation Scheme	
31	NN935211	☒	Multisequencing a Single Instruction Stream - Concurrent Execution on Alternative Decoder Paths	
32	NB910349	☒	Threaded Code Design for SET Operations	
33	WO 2004016026 A	☒	Application level message carrying method for cellular network, involves delivering encapsulated message to receiver application process by transmitting signaling message	
34	US 20040013116 A	☒	Mobile IP functionality providing method for non mobile IP capable mobile node, involves discovering address of home agent by switching device and advertising address as care-of-address with respect to former address	
35	US 20040010788 A	☒	Hardware context allocation method for microprocessor, involves permanently allocating hardware content to specific virtual machines based on their resource requirements	
36	WO 2003107616 A	☒	Internet protocol headers decompressor node for initializing headers compression, has application module that decompresses application related Internet protocol packets exchanged with one of nodes using decompression context	
37	US 20030229710 A	☒	Incoming data stream matching method for use in broadband data networking equipment, involves determining potential pattern matches of contexts generated by dividing incoming data stream, in database of known signatures	
38	US 20030217174 A	☒	Internet protocol session initiating method, involves receiving session request containing host contact and device identifier with device address that is obtained by activation of packet data protocol context	
39	US 6650877 B	☒	Physical parameter sensing method for use with radio broadcast context, involves identifying data location for physical parameter transmitted from listener terminal, and returning identified location to listener	
40	WO 2003092314 A	☒	Internet protocol session relocation support method, involves determining potential next network node applicability to relocate Internet protocol session by capability information on set of capabilities	
41	CN 1450750 A	☒	Method for realizing multi-casting roam	
42	US 6625635 B	☒	Threads scheduling method for multi-threaded data processing system, involves allocating prespecified number of instructions to instruction counter which counts execution of instruction by threads sequentially	

CACHE COHERENCY PROTOCOL IN WHICH A LOAD INSTRUCTION HINT BIT IS EMPLOYED TO INDICATE DEALLOCATION OF A MODIFIED CACHE LINE SUPPLIED BY INTERVENTION

CROSS-REFERENCES TO RELATED APPLICATIONS

The present invention is related to the following applications filed concurrently with this application: U.S. patent application Ser. No. 09/437,179 entitled "HIGH PERFORMANCE MULTIPROCESSOR SYSTEM WITH MODIFIED-UNSOLICITED CACHE STATE"; U.S. patent application Ser. No. 09/437,178 entitled "MULTIPROCESSOR SYSTEM BUS PROTOCOL WITH COMMAND AND SNOOP RESPONSES FOR MODIFIED-UNSOLICITED CACHE STATE"; U.S. patent application Ser. No. 09/437,177 entitled "MODIFIED-UNSOLICITED CACHE STATE WITH DYNAMIC HARDWARE/SOFTWARE CONTROLLABLE MECHANISM TO OPTIMIZE FOR LOAD IMBALANCE"; U.S. patent application Ser. No. 09/437,180 entitled "PROTOCOL FOR TRANSFERRING MODIFIED-UNSOLICITED STATE DURING DATA INTERVENTION"; U.S. patent application Ser. No. 09/437,181 entitled "CACHE ALLOCATION MECHANISM FOR MODIFIED-UNSOLICITED CACHE STATES".

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention generally relates to computer systems, and more particularly to a cache coherency protocol which provides novel mechanisms for handling data in a multi-processor computing system, resulting in improved cache usage and the reduction of bus traffic between horizontal caches when a value contained in one cache is modified after being retrieved, in a previously modified form, from a cache of an adjacent processing unit.

2. Description of Related Art

The basic structure of a conventional multi-processor computer system **10** is shown in FIG. **1**. Computer system **10** has several processing units, two of which **12***a* and **12***b* are depicted, which are connected to various peripheral devices, including input/output (I/O) devices **14** (such as a display monitor, keyboard, graphical pointer (mouse), and a permanent storage device or hard disk), memory device **16** (such as random access memory or RAM) that is used by the processing units to carry out program instructions, and firmware **18** whose primary purpose is to seek out and load an operating system from one of the peripherals (usually the permanent memory device) whenever the computer is first turned on. Processing units **12***a* and **12***b* communicate with the peripheral devices by various means, including a generalized interconnect or bus **20**, or direct memory access channels (not shown). Computer system **10** may have many additional components which are not shown, such as serial, parallel, and universal system bus (USB) ports for connection to, e.g., modems, printers or scanners. There are other components that might be used in conjunction with those shown in the block diagram of FIG. **1**; for example, a display adapter might be used to control a video display monitor, a memory controller can be used to access memory **16**, etc. The computer can also have more than two processing units.

In a symmetric multi-processor (SMP) computer, all of the processing units are generally identical, that is, they all use a common set or subset of instructions and protocols to operate, and generally have the same architecture. A typical architecture is shown in FIG. **1**. A processing unit includes a processor core **22** having a plurality of registers and execution units, which carry out program instructions in order to operate the computer. An exemplary processing unit includes the PowerPC™ processor marketed by International Business Machines Corp. The processing unit can also have one or more caches, such as an instruction cache **24** and a data cache **26**, which are implemented using high speed memory devices. Caches are commonly used to temporarily store values that might be repeatedly accessed by a processor, in order to speed up processing by avoiding the additional latency of loading the values from memory **16**. These caches are referred to as "on-board" when they are integrally packaged with the processor core on a single integrated chip **28**. Each cache is associated with a cache controller (not shown) that manages the transfer of data and instructions between the processor core and the cache memory.

A processing unit can include additional caches, such as cache **30**, which is referred to as a level **2** (L2) cache since it supports the on-board (level **1**) caches **24** and **26**. In other words, cache **30** acts as an intermediary between memory **16** and the on-board caches, and can store a much larger amount of information (instructions and data) than the on-board caches can, but at a longer access penalty. For example, cache **30** may be a chip having a storage capacity of 512 kilobytes, while the processor may be an IBM PowerPC™ 604-series processor having on-board caches with 64 kilobytes of total storage. Cache **30** is connected to bus **20**, and all loading of information from memory **16** into processor core **22** must come through cache **30**. Although FIG. **1** depicts only a two-level cache hierarchy, multi-level cache hierarchies can be provided where there are many levels (L3, L4, etc.) of serially connected caches.

In a multi-level cache, if a copy of a value is in every level of the cache, the cache hierarchy is referred to as being "inclusive." It is not necessary, however, to keep a copy of each value in the lower levels, and an inclusivity bit field may be added to the caches to indicate whether or not the cache is inclusive. For example, a three-level cache structure might provide an L3 cache which was not inclusive, such that a value residing in the L2 cache might not be present in the L3 cache. In this example, if an L2 cache issues a read command for a value that is not present in any of the caches of that processing unit, it can be passed to that L2 cache without (necessarily) loading it into the L3 cache.

In an SMP computer, it is important to provide a coherent memory system, that is, to cause write operations to each individual memory location to be serialized in some order for all processors. By way of example, assume a location in memory is modified by a sequence of write operations to take on the values: 1, 2, 3, 4. In a cache coherent system, all processors will observe the writes to a given location to take place in the order shown. However, it is possible for a processing element to miss a write to the memory location. A given processing element reading the memory location could see the sequence 1, 3, 4, missing the update to the value 2. A system that implements these properties is said to be "coherent". Nearly all coherency protocols operate only to the granularity of the size of a cache block. That is to say, the coherency protocol controls the movement of and write permissions for operand data or instructions on a cache block basis, and not separately for each individual memory location.

There are a number of protocols and techniques for achieving cache coherence that are known to those skilled in

	Document ID	U	Title	Current OR
43	US 20030169704 A	☒	Mobile communication terminal, has packet data protocol context management reestablishing context when terminal moves to service area after timer completes measuring full measuring time value before update timer	
44	DE 10205907 A	☒	Changing internet protocol access connections to remote IP unit establishes second connection between mobile client and server, using different addresses	
45	EP 1326408 A	☒	Enabling several virtual servers to participate in several private network address spaces by applying first IP space ID to translation procedures, enabling selection of current virtual server context	
46	US 20030126059 A	☒	IP (intellectual property) brokering system for use in anti-terrorism prevention and damage control, has IP brokering unit and communication unit for processing customer requirements and deliverable IP between customers and IP agent	
47	WO 2003055170 A	☒	Handling end-to-end Internet business transactions by generating session context with allocated IP address and user ID for validation by transaction manager	
48	KR 2003054980 A	☒	Incoming service method in mobile communication packet network	
49	US 20030112794 A	☒	General packet radio service telecommunications system has port assignment module which sequentially assigns multiple internet protocol addresses to same transmission control protocol ports	
50	US 20030097548 A	☒	Processing system e.g. pipelined computer processing system, selects address and instructions from respective context registers simultaneously for each cycle of execution of processor unit	
51	US 20030038961 A	☒	Client message structure for data processing system, includes formats having several message header segments for identifying message, and its property, that are produced in accordance with established protocol	
52	WO 2003013165 A	☒	Telecommunications arrangement for mobile packet switched communication system in which support node is interrogated to identify resource wasting hanging PDP contexts	
53	KR 2003060936 A	☒	Wap push service method in 3gpp imt-2000 packet network	
54	US 20020199179 A	☒	Code execution method for copyright protection of patent document, involves selecting entry corresponding to trigger instruction, for executing corresponding auxiliary code	
55	US 20020150082 A	☒	Voice call processing system for IP telephony, has real time streaming engine with higher priority and normal priority threads for transmitting audio data and silence data to VoIP gateway respectively	
56	KR 2002077755 A	☒	Nms platform using multiple thread	
57	WO 2002548 11 A	☒	Position determination for terminal equipment e.g. mobile telephone in GPRS system that does not use GPS or other costly technologies	
58	US 20020087843 A	☒	Simultaneous multithreaded processor system delivers instruction pointers for active and inactive threads to processor logic and storage element, respectively	
59	US 20020078135 A	☒	Data moving method in application layer proxy, involves storing address pointer in send queue of communication connection, so as to move data from one communication connection to another connection	

the art. All of these mechanisms for maintaining coherency require that the protocols allow only one processor to have a "permission" that allows a write operation to a given memory location (cache block) at any given point in time. As a consequence of this requirement, whenever a processing element attempts to write to a memory location, it must first inform all other processing elements of its desire to write the location and receive permission from all other processing elements to carry out the write.

To implement cache coherency in a system, the processors communicate over a common generalized interconnect (i.e., bus 20). The processors pass messages over the interconnect indicating their desire to read from or write to memory locations. When an operation is placed on the interconnect, all of the other processors "snoop" (monitor) this operation and decide if the state of their caches can allow the requested operation to proceed and, if so, under what conditions. There are several bus transactions that require snooping and follow-up action to honor the bus transactions and maintain memory coherency. The snooping operation is triggered by the receipt of a qualified snoop request, generated by the assertion of certain bus signals. Instruction processing is interrupted only when a snoop hit occurs and the snoop state machine determines that an additional cache snoop is required to resolve the coherency of the offended sector.

This communication is necessary because, in systems with caches, the most recent valid copy of a given block of memory may have moved from the system memory **16** to one or more of the caches in the system (as mentioned above). If a processor (say **12***a*) attempts to access a memory location not present within its cache hierarchy, the correct version of the block, which contains the actual (current) value for the memory location, may either be in the system memory **16** or in one of more of the caches in another processing unit, e.g. processing unit **12***b*. If the correct version is in one or more of the other caches in the system, it is necessary to obtain the correct value from the cache(s) in the system instead of system memory.

For example, consider a processor, say **12***a*, attempting to read a location in memory. It first polls its own L1 cache (**24** or **26**). If the block is not present in the L1 cache, the request is forwarded to the L2 cache (**30**). If the block is not present in the L2 cache, the request is forwarded on to lower cache levels, e.g., the L3 cache. If the block is not present in the lower level caches, the request is then presented on the generalized interconnect (**20**) to be serviced. Once an operation has been placed on the generalized interconnect, all other processing units snoop the operation and determine if the block is present in their caches. If a given processing unit has the block requested by processing unit in its L1 cache, and the value in that block is modified, and any lower level caches also have copies of the block, then their copies are stale, since the copy in the processors cache is modified. Therefore, when the lowest level cache (e.g., L3) of the processing unit snoops the read operation, it will determine that the block requested is present and modified in a higher level cache. When this occurs with an inline cache structure, the L3 cache places a message on the generalized interconnect informing the processing unit that it must "retry" it's operation again at a later time, because the actual value of the memory location is in the L1 cache at the top of the memory hierarchy and must be retrieved to make it available to service the read request of the initiating processing unit.

Once the request from an initiating processing unit has been retried, the L3 cache begins a process to retrieve the modified value from the L1 cache and make it available at the L3 cache, main memory or both, depending on the exact details of the implementation. To retrieve the block from the higher level caches, the L3 cache sends messages through the inter-cache connections to the higher level caches, requesting that the block be retrieved. These messages propagate up the processing unit hierarchy until they reach the L1 cache and cause the block to be moved down the hierarchy to the lowest level (L3 or main memory) to be able to service the request from the initiating processing unit.

The initiating processing unit eventually re-presents the read request on the generalized interconnect. At this point, however, the modified value has been retrieved from the L1 cache of a processing unit and placed into system memory, and the read request from the initiating processor will be satisfied. The scenario just described is commonly referred to as a "snoop push". A read request is snooped on the generalized interconnect which causes the processing unit to "push" the block to the bottom of the hierarchy to satisfy the read request made by the initiating processing unit.

Thus, when a processor wishes to read or write a block, it must communicate that desire with the other processing units in the system in order to maintain cache coherence. To achieve this, the cache coherence protocol associates with each block in each level of the cache hierarchy, a status indicator indicating the current "state" of the block. The state information is used to allow certain optimizations in the coherency protocol that reduce message traffic on the generalized interconnect and the inter-cache connections. As one example of this mechanism, when a processing unit executes a read it receives a message indicating whether or not the read must be retried (i.e., reissued later). If the read operation is not retried, the message usually also includes information allowing the processing unit to determine if any other processing unit also has a still active copy of the block (this is accomplished by having the other lowest level caches give a "shared" or "not shared" indication for any read they do not retry). Therefore, a processing unit can determine whether any other processor in the system has a copy of the block. If no other processing unit has an active copy of the block, the reading processing unit marks the state of the block as "exclusive". If a block is marked exclusive it is permissible to allow the processing unit to later write the block without first communicating with other processing units in the system because no other processing unit has a copy of the block. Therefore, it is possible for a processor to read or write a location without first communicating this intention onto the interconnection, but only where the coherency protocol has ensured that no other processor has an interest in the block.

The foregoing cache coherency technique is implemented in a specific protocol referred to as "MESI," and illustrated in FIG. 2. In this protocol, a cache block can be in one of four states, "M" (Modified), "E" (Exclusive), "S" (Shared) or "I" (Invalid). Under the MESI protocol, each cache entry (e.g., a 32-byte sector) has two additional bits which indicate the state of the entry, out of the four possible states. Depending upon the initial state of the entry and the type of access sought by the requesting processor, the state may be changed, and a particular state is set for the entry in the requesting processor's cache. For example, when a sector is in the Modified state, the addressed sector is valid only in the cache having the modified sector, and the modified value has not been written back to system memory. When a sector is Exclusive, it is present only in the noted sector, and is consistent with system memory. If a sector is Shared, it is valid in that cache and in at least one other cache, all of the shared sectors being consistent with system memory. Finally, when a sector is Invalid, it indicates that the

	Document ID	U	Title	Current OR
60	JP 2002141931 A	☒	Router device for internet, has context ID rewriting unit which modifies similar context ID owned by several packets using unique number assigned to each address	
61	WO 200232170 A	☒	Address de-registration in Internet protocol multimedia networks of addresses assigned to users in general packet radio service and Internet protocol multimedia networks for providing application services	
62	WO 200232084 A	☒	Address de-registration from Internet protocol multimedia network to de-register addresses assigned to users using gating network element to detect communication channel deactivation	
63	WO 200121397 A	☒	Header compression context control during handover in mobile data communication networks that prevents errors and communications loss during handover	
64	GB 2362789 A	☒	Real-time bit pattern search system in transport packet scheme, e.g. IP, ATM, has tracking unit which tracks search context information for each data stream portion	
65	US 20020012433 A	☒	Authentication method for a mobile node to a packet data network, e.g. IP, using shared secret key of the mobile and the network authentication center	
66	US 6295600 B	☒	Microprocessor for executing multithreaded program in computer, fetches new thread based on stored address in response to cache miss indication signal	
67	US 6295557 B	☒	Internet traffic simulator for testing ability of site to handle transport control protocol connections, retrieves information from transport control protocol connection and records statistics related to the information	
68	EP 1133140 A	☒	Multi-service communication system has PSTN gateway permitting unidirectional and bidirectional communication between PSTN interface and IP interface	
69	KR 2001063804 A	☒	Method for maintaining object consistency in distributed-object system	
70	KR 2001048669 A	☒	Method for interfacing application program of mobile event information	
71	KR 2001035104 A	☒	Method for registering mobile ip having minimum band in asynchronous mobile communication system	
72	WO 200131881 A	☒	Internet protocol header compression initiation in packet switched network, involves initiating compression, when full header packet with modified header and inserted routing header is received by destination router	
73	KR 2001007695 A	☒	Chatting service method and system using virtual chatting server	
74	WO 20015171 A	☒	Access network protocol context management method for use in access network, involves monitoring macro mobility registration at gateway node to determine necessity of access network protocol context	
75	KR 2001002485 A	☒	Multi-thread microprocessor for instruction fetch	
76	US 6151569 A	☒	User access reestablishing method to programs in computer system involves initiating multitasked execution of identified program with modified context registers of identified program	
77	WO 200064203 A	☒	Multimedia related information transmission involves exchanging information between transport protocol layers in terminal and network device arrangements via octet stream and network transmission protocol layers	

addressed sector is not resident in the cache. As seen in FIG. 2, if a sector is in any of the Modified, Shared or Invalid states, it can move between the states depending upon the particular bus transaction. While a sector in an Exclusive state can move to any other state, a sector can only become Exclusive if it is first Invalid.

A further improvement in accessing cache blocks can be achieved using the cache coherency protocol. This improvement, referred to as "intervention," allows a cache having control over a memory block to provide the data in that block directly to another cache requesting the value (for a read-type operation), in other words, bypassing the need to write the data to system memory and then have the requesting processor read it back again from memory. Intervention can generally be performed only by a cache having the value in a block whose state is Modified or Exclusive. In both of these states, there is only one cache block that has a valid copy of the value, so it is a simple matter to source (write) the value over the bus 20 without the necessity of first writing it to system memory. The intervention procedure thus speeds up processing by avoiding the longer process of writing to and reading from system memory (which actually involves three bus operations and two memory operations). This procedure not only results in better latency, but also increased bus bandwidth.

There are many variations of the MESI protocol. The tagged ("T") state is used to identify a cache block which is inconsistent with system memory (i.e., modified) and is further responsible for writing the correct (current) value to memory upon deallocation (or to pass on the tag to another cache block during intervention). The T state can be used to share a modified value, by marking one of the sharing blocks as (temporarily) responsible for maintaining a valid copy of the value. The recently read ("R") state can be used to allow intervention when the value is unmodified but shared among many caches, so as to conveniently mark a single one of the sharing caches as being responsible for intervention. The hover ("H") state allows a cache line to maintain an address in the directory even though the corresponding value in the cache entry array is an invalid copy, so that it can snoop the correct value for its processing unit if the value happens to be broadcast as part of an intervention between the caches of two other processing units.

In many instances, a value that is modified by a store operation, and thus present in a cache of the particular processing unit that issued the store operation, will subsequently be modified by another processing unit before the cache evicts the modified value. When one cache issues a read request and another cache contains a modified value corresponding to the requested address, the other cache typically will intervene the value, copy the line back to system memory, and switch its cache line from the modified (M) cache coherency state to the shared (S) state. If, however, the requesting cache quickly follows up the read request with a write operation to the same address, then the other (intervening) cache must deallocate the corresponding cache line, i.e., switch its state from shared to invalid (I). This follow-up write operation accordingly requires a bus transaction (e.g., a DClaim instruction) to inform the other cache that it must deallocate the line. This bus transaction to invalidate a cache line may take as long as 100 processor cycles to complete, which can ultimately delay processing. It would, therefore, be desirable to devise a method of maintaining cache coherency which allowed a requesting cache to modify a value provided via intervention in response to a read request, without having to issue a bus transaction, in order to decrease bus traffic and speed up processing. It would be further advantageous if the method could be selectively implemented for those applications which tended to perform such modifications quickly after reading the value from another cache.

SUMMARY OF THE INVENTION

It is therefore one object of the present invention to provide an improved method of maintaining cache coherency in a multi-processor computer system.

It is another object of the present invention to provide such a method that allows a cache to avoid unnecessary bus transactions issued to other horizontal caches.

It is yet another object of the present invention to provide a computer system which makes more efficient use of a cache having a cache line replacement/victimization algorithm.

The foregoing objects are achieved in a method of maintaining cache coherency, comprising the steps of issuing a load instruction from a first cache of a first processing unit, and sending a hint bit with the load instruction wherein the hint bit is provided as part of an instruction set architecture of the computer system and indicates whether a second cache of a second processing unit should deallocate a corresponding cache line upon sourcing a modified value to the first cache responsive to the load instruction. The modified value is stored into the cache line of the second cache, prior to said issuing step, and a first coherency state (the conventional M state) is assigned to the cache line of the second cache indicating that the cache line contains the modified value. When the hint bit is set to indicate that the second cache should deallocate the cache line, the modified value is sourced from the cache line of the second cache to a cache line of the first cache, the cache line of the second cache deallocated, and a second coherency state (the modified-unsolicited, or M_U state) is assigned to the cache line of the first cache indicating that the cache line of the first cache contains the value as modified by another processing unit and that the modified value has not been written to a system memory device.

The above as well as additional objectives, features, and advantages of the present invention will become apparent in the following detailed written description.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, however, as well as a preferred mode of use, further objectives, and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

FIG. 1 is a block diagram of a prior-art multi-processor computer system;

FIG. 2 is a state diagram depicting a prior art cache coherency protocol (MESI);

FIG. 3 is a state diagram depicting the cache coherency protocol of the present invention, providing a novel coherency state (M_U) to allow a cache to modify a value, in response to a store operation, without issuing a bus transaction to another cache which previously intervened the requested value;

FIG. 4 is a block diagram illustrating one embodiment of the invention within a multi-processor computing system having at least three processing units with associated caches; and

	Document ID	U	Title	Current OR
78	JP 2000112772 A	☒	Data processing system of digital computer, has decoder which decodes instruction in sequence and detects context call instruction	
79	EP 942365 A	☒	Instruction counter for triggering context controller switch to next background task	
80	US 5889848 A	☒	Intelligent peripheral operating method for telecommunication intelligent network	
81	US 5710923 A	☒	Communicating active messages among nodes of parallel processing computer system - performing procedure identified by instruction pointer on data structure identified by frame pointer in accordance with micro-thread, so that procedure uses Local Parameters pointer to locate additional parameters associated with procedure	
82	US 5754830 A	☒	Web/emulator server for providing persistent connection between client and legacy host system - in which applet code is downloaded to client system in response to receiving uniform resource locator associated with legacy host system	
83	JP 09185515 A	☐	System timer management method for IPS - by establishing interruption period of system timer using period establishing circuit based on system load condition detected by load detector according to interruption context	
84	JP 09034848 A	☐	Threading control method for distributed IPS - involves updating memory area containing mutual relationship between threads based on thread which accesses distributed virtual share memory	
85	US 5459845 A	☐	Instruction pipeline sequencing method - in which state information of instruction travels through pipe stages until instruction execution is completed	
86	EP 580109 A	☐	Data processor with disc access in RISC DSP - determines prefetch and poststore addresses according to rule determined by context register with index indicative of this address stored in index register, all together forming a streamer between CPU and data memory	
87	US 5499349 A	☐	Parallel multi thread data processing system - has instruction pointer and frame pointer which identify instructions to be executed and operated on	
88	EP 134386 A	☐	Object code execution appts. from high-level language source - includes host computer with emulator for low-level operations and language processor for high-level	
89	SU 612692 A	☐	Conical crusher with slit regulator adjusting ring - has bracket on lock nut carrying lock connected to casing to simplify design	

FIG. 5 is a chart illustrating the logic flow according to one implementation of the present invention.

DESCRIPTION OF AN ILLUSTRATIVE EMBODIMENT

With reference now to FIG. 3, there is depicted a state diagram of one embodiment of the cache coherency protocol of the present invention. This protocol is similar to the prior art MESI protocol of FIG. 2, in that it includes the same four prior states (Modified, Exclusive, Shared and Invalid), but it also includes a new "M_U" state (Modified-Unsolicited). The M_U state indicates that: (1) the subject value has been modified (i.e., is not currently consistent with system memory); (2) the value was modified by another processing unit (i.e., not by the processing unit associated with the cache that currently contains the value in the M_U state); and (3) the value is exclusively held (i.e., it is not present in any other horizontally adjacent caches). Because the value is exclusively held, it may be modified in that cache without the necessity of issuing a bus transaction to other horizontal caches in the memory hierarchy.

If a processor core executes a load operation and its cache does not currently contain a valid copy of the requested value, the cache passes the load operation to the remainder of the memory hierarchy to obtain the value. If the value is not present in any cache of the computer system, it is provided by system memory. According to one implementation of the present invention, however, if the value is currently contained in a cache associated with another processing unit, and has already been modified by that processing unit (i.e., is currently in the M state), then the value is retrieved from that cache (via intervention), loaded into a cache of the requesting processor, and assigned the M_U state. The intervening cache immediately deallocates the corresponding cache line. Thus, when the same processor core issues a subsequent store instruction to the same address, the M_U state provides an indication to its cache that the value is not currently contained in any other horizontally adjacent caches within the memory hierarchy. Accordingly, the cache is free to store the modified value, and switch the state of its cache line from M_U to M without issuing any bus transactions, as there is no need to deallocate any cache line(s) in the other cache(s).

In contrast, in the prior art, it would be necessary to issue a deallocation request (e.g., DClaim operation) to the intervening cache which would still contain the value. If the standard MESI protocol were in use, upon intervention from the responding cache, both caches would contain the value in the shared (S) state—the value would also be copied back to system memory during intervention. If the alternative T-MESI protocol were implemented, the value would not be written back to system memory, but the responding cache would still hold the value in the S state, and the requesting cache would assign the T state to the cache line. In either case, when the processor core of the requesting cache issued the follow-up write instruction, it would be necessary to deallocate the line held in the S state in the other cache. This deallocation operation would increase bus traffic and delay processing.

In a preferred implementation of the present invention, a flag (single-bit field) is sent with the read request to indicate that the requesting cache is capable of utilizing the M_U state (i.e., owning previously modified data). Accordingly, the snoop response from the intervening cache can specifically instruct the requesting cache to utilize that state, in which case the intervening cache will switch from the modified state to the invalid state. This approach allows the use of the modified-unsolicited state to be optional, as the intervening cache might alternatively send a conventional snoop response, e.g., instructing the requesting cache to assign the shared or tagged state to the cache block. Alternatively, the master (intervening cache) could send a snoop response instructing the requesting cache to simply assign the conventional modified state to the block.

Use of this flag may be programmable via either software or hardware. Software that is visible to the computer's operating system may make a determination that it is desirable to use the M_U state (based on the particular program application running), and instruct the operating system to enter an operating mode in which the flag is appropriately set. Hardware may alternatively be provided to dynamically monitor cache usage; for example, the modified-unsolicited state might be disabled if the monitor determines that many castouts and/or modified intervention requests have occurred on M_U cache lines. Each processing unit could have its own monitor, or a system-wide monitor could be provided (e.g., as part of the memory controller). If the flag sent with the request indicates that the M_U state is to be excluded, then conventional intervention occurs.

The programmability feature could further be made visible to a user by adding a hint bit to the load instruction, as part of the instruction set architecture. The instruction set architecture (ISA) for the processor core (e.g., the ISA of a PowerPC™ 630 processor) could be extended to allow software, via the compiler program, to directly indicate the desirability of using the M_U state for a particular set of load instructions using the hint bit. The availability of the feature as part of the ISA is particularly advantageous since it can be dynamically implemented with each individual instruction.

In an alternative implementation of the present invention, a flag can be sent with the intervention data to indicate that the M_U state may be used. In other words, the requesting cache passes the read operation to the remainder of the memory hierarchy in a conventional fashion, but the intervening cache sends the requested value with a flag indicating that the requesting cache can (should) assign the M_U state to the cache line. This flag thus indicates that the requested value has been supplied via intervention rather than from system memory, and that the line in the intervening cache has already been deallocated. This variation differs from the above-explained implementation since it is not the snoop response that instructs the requesting cache to use the M_U state, but rather the indicating flag is supplied along with the requested data.

In another refinement of the present invention, the eviction/victimization algorithm used by the cache may be modified to further enhance cache efficiency. If a cache miss occurs, and if all of the blocks in a particular congruence class already have valid copies of memory blocks, then one of the cache blocks must be selected for victimization. This selection, in the prior art, is typically performed using an array of bits that indicate which of the blocks is the least-recently used (LRU). For example, there are three LRU bits per block for an 8-way set associative cache. The LRU bits from each block in the class are provided as inputs to a decoder having an 8-bit output to indicate which of the blocks is to be victimized. Several LRU hardware solutions have been implemented to provide "fair" information replacement. Fair replacement, however, often does not provide the best application performance. For some applications, it may be preferable to evict a cache line that is in the M_U state, while for others it may be preferable to

	Document ID	U	Title	Current OR
1	JP 20022 97414 A	☐	SYSTEM SIMULATOR, SIMULATION METHOD AND SIMULATION PROGRAM	
2	JP 20020 73374 A	☒	INTERRUPT SIMULATION METHOD AND DEVICE	
3	JP 20022 97414 A	☒	SYSTEM SIMULATOR, SIMULATION METHOD AND SIMULATION PROGRAM	
4	JP 20020 73374 A	☒	INTERRUPT SIMULATION METHOD AND DEVICE	
5	JP 09190 348 A	☒	INSTRUCTION PREFETCH BUFFER CONTROL METHOD AND DEVICE THEREFOR AND INSTRUCTION PREFETCH BUFFER FLUSH METHOD	
6	JP 08227 364 A	☒	METHOD AND DEVICE FOR EXECUTING SEQUENTIAL MULTITHREAD	
7	JP 03137 702 A	☒	NUMERICALLY CONTROLLED THREAD CUTTING DEVICE	
8	DE 10110 504 A1	☒	Use of threads in connection with processor and accessible data by transferring execution control to next thread in queue if first thread is blocked	
9	WO 97457 95 A1	☒	PARALLEL PROCESSOR WITH REDUNDANCY OF PROCESSOR PAIRS	
10	WO 97076 21 A1	☒	PARALLEL EXECUTION OF REQUESTS IN OSI AGENTS	
11	EP 73547 5 A2	☒	Method and apparatus for managing objects in a distributed object operating environment	
12	EP 61736 1 A2	☒	Scheduling method and apparatus for a communication network.	
13	US 20040 00302 3 A	☒	Computer system processor selection method for loading processing thread, involves overwriting candidate/volunteer processor information based on comparison of load between candidate/volunteer processor	
14	US 66256 35 B	☒	Threads scheduling method for multi-threaded data processing system, involves allocating prespecified number of instructions to instruction counter which counts execution of instruction by threads sequentially	
15	US 20030 04651 7 A	☒	Multithreading apparatus for computer processor pipeline, has control unit statically scheduled to execute multiple threads in round robin succession to eliminate need for communication between pipeline stages	
16	JP 20030 52687 A	☒	Multislice/cone beam computerized tomography devices undergo several interrelated image data corrective procedures of varying relevance/emphasis	
17	US 20020 10800 3 A	☒	Data controller for peripheral device, generates transfer extend entries/retrieval for data transfer, automatically	
18	WO 20025 4246 A	☒	Method for testing and monitoring parallel sequentially executed process sections, or threads, in an existing software development system by extension of thread handlers to include event tracers	
19	US 63341 71 B	☒	Write-combining device for uncacheable stores in computer system, receives two stores for write-combining, if both stores are sequentially executed from same thread	

never victimize M_U data, if possible. Accordingly, the LRU bits may be set for M_U data depending upon the particular application. For example, when a value is loaded into a prior art cache, its LRU bits would be set to indicate that it is the most-recently used (MRU) block, but the present invention can override this setting for M_U data, to instead set the bits to indicate that the block is the least-recently used. Conversely, an LRU decoder can be designed such that M_U data is never evicted (unless all blocks in the congruence class are in the M_U state).

FIG. 4 illustrates one embodiment of a multi-processor computer system in which the present invention can be practiced. The invention could be applied to computer systems having new hardware components not shown in FIG. 4, or having other interconnection architectures, so those skilled in the art will appreciate that the present invention is not limited to the generalized system shown in that figure. In this embodiment, the hierarchy includes a system memory device 40 connected via a system bus 42 to a plurality of processing units, 44*a*, 44*b* and 44*c*. Each of these processing units has at least one cache 46*a*, 46*b* and 46*c* (the invention could be applied in multi-level cache hierarchies as well). As seen in FIG. 4, when a cache line is read from cache 46*b* which already has the line in the modified (M) state, the line goes from invalid (I) to modified-unsolicited (M_U) in the requesting cache 46*a*, and the corresponding line in intervening cache 46*b* is immediately deallocated (goes from M to I). Thereafter, if a store operation were to hit cache 46*a*, the line would go from modified-unsolicited to modified (M) (not illustrated in FIG. 4), without any bus transactions between cache 46*a* and cache 46*b*. If, however, no store operation occurs prior to a load request from cache 46*c* then, as illustrated in FIG. 4, cache 46*a* can intervene the value and switch its state to shared (S), with the value being written to system memory 40 as well. The line in cache 46*c* is also assigned the shared state. As indicated by the parentheses in FIG. 4, the line in cache 46*a* could switch to the tagged (T) state if the value is not to be immediately written back to system memory 40.

The logic flow for this implementation is shown in FIG. 5. First, a line is modified in one of the caches (50). A read request is then issued from a second cache, which may be issued with the flag discussed above (52). A snoop response is formulated to indicate that the line can be loaded in the M_U state in the second cache (54). The cache line is then loaded into the second cache, and set to the M_U state (56); the line in the first cache switches from the M state to the I state. Thereafter, if the line is modified (58), then the second cache can switch to the M state with no bus transaction (60). Whether the line is modified or not, if a read request is issued from a third cache (62), the line in the second cache switches to the S (or T) state (64) when it intervenes the data to the third cache.

Although the invention has been described with reference to specific embodiments, this description is not meant to be construed in a limiting sense. Various modifications of the disclosed embodiments, as well as alternative embodiments of the invention, will become apparent to persons skilled in the art upon reference to the description of the invention. It is therefore contemplated that such modifications can be made without departing from the spirit or scope of the present invention as defined in the appended claims.

What is claimed is:

1. A method of maintaining cache coherency in a computer system having a plurality of processing units, each processing unit having at least one cache, comprising the steps of:

compiling a program having a plurality of instructions including at least one load instruction which provides a hint bit to indicate whether a cache containing a requested value in modified form should deallocate a corresponding cache line upon sourcing the requested value by intervention to a cache of another processing unit;

in response to execution of the load instruction, issuing a read request with the hint bit from a first cache of a first processing unit to a second cache of a second processing unit;

in response to receiving the requested value in a cache line of the first cache assigning a first coherency state to the cache line of the first cache indicating that the cache line of the first cache contains the value as modified by another processing unit and that the modified value has not been written to a system memory device.

2. The method of claim 1 further comprising the steps of:

storing the modified value into the cache line of the second cache, prior to said issuing step; and

assigning a second coherency state to the cache line of the second cache indicating that the cache line contains the modified value.

3. The method of claim 2 further comprising the steps of:

setting the hint bit to indicate that the second cache should deallocate the cache line;

sourcing the modified value from the cache line of the second cache to a cache line of the first cache; and

deallocating the cache line of the second cache.

4. The method of claim 3 further comprising the step of sending a snoop response from the second cache instructing the first cache to utilize the first coherency state.

5. The method of claim 3 further comprising the steps of:

issuing a store operation for an address associated with the modified value from the first processing unit, after said sourcing step; and

modifying the value in the cache line of the first cache in response to the store operation, wherein said modifying step occurs without issuing any bus transaction from the first cache.

6. The method of claim 5 further comprising the step of assigning the second coherency state to the cache line of the first cache in response to said modifying step.

7. The method of claim 3 further comprising the step of modifying a plurality of victimization priority bits associated with the cache line of the first cache in response to said step of assigning the first coherency state.

8. A computer system comprising:

a system memory device;

a plurality of processing units each having a cache, each of said processing units adapted to execute a load instruction which provides a hint bit to indicate whether a cache containing a requested value in modified form should deallocate a cache line containing the requested value upon sourcing the requested value by intervention to a cache of another processing unit;

bus means for interconnecting said processing units and said system memory device; and

cache coherency means, responsive to execution of the load instruction, for issuing a read request with the hint bit from a first cache of a first processing unit to a second cache of a second processing unit and, responsive to receiving the requested value in a cache line of the first cache, for assigning a first coherency state to said cache line of said first cache indicating that said

	Document ID	U	Title	Current OR
20	WO 20018 8713 A	☒	Garbage collection process for computer memory pushes identifiers onto thread work queue and pops task identifiers from thread queue ends	
21	DE 10110 504 A	☒	Use of threads in connection with processor and accessible data by transferring execution control to next thread in queue if first thread is blocked	
22	US 20020 00266 7 A	☒	Embedded processor architecture for enabling multithreading, invokes zero-time context switches between end and beginning of program instruction execution states in threads	
23	US 62431 07 B	☒	Graphics processor system performance optimizing method involves moving master thread between slave threads of respective processors to cause each processor to execute its graphics pipeline	
24	US 62054 94 B	☒	Interface controller for disk drive, includes command queuing engine to generate data transfer descriptor for each initiator command and to form thread of sequential data transfers	
25	RD 43314 9 A	☒	Improving performance score of multi-thread sequential read	
26	JP 10011 301 A	☒	Multitask processing apparatus used in microprocessor - has memory for context provided separately from internal bus to store each context by which tasks are to be performed sequentially	
27	US 56969 15 A	☒	Computer controlled display system associated routes user action controlling - determining if computer controlled display system is in first context then automatically sequentially executes each routine of first number of routines	
28	EP 90165 9 B	☒	Parallel processor with redundancy of processor pairs - with direct pairing between processors of separate memory buses, such that two tightly coupled processors can reciprocally synchronise themselves and share internal register files	
29	US 58988 32 A	☒	Organising CMIP request execution in OSI environment - providing sub threads within each main thread for simultaneous processing of multiple CMIP requests	
30	EP 83935 0 B	☒	Transaction synchronisation method for processing system having agents and coordinator - sending vote indicating availability to commit from each agent to coordinator, and determining commit or back-out decision by coordinator when all votes are received	
31	US 54780 51 A	☒	Mfg. elongated belt having sequentially located plastic components - by moulding component part and guide means with location determining means interconnected by cross-piece onto thread of compatible material at moulding station.	
32	EP 64908 7 A	☒	Computer system for software function selection - has filtering mechanism for finding and selecting collection of software functions useful for user from collection of available software provided by software developers	
33	EP 64448 6 A	☒	Management system for data access in computer disc memory system - groups together large data access tasks against partitions onto physical disc to be read from or written to, and executes tasks serially in accordance with disc task list	
34	US 53902 81 A	☐	Deducing user intent and providing computer implemented services for pen-based computer - comparing new events with events stored in database, collecting into set, comparing to intent templates, and selecting best-guess hypothesis	

cache line of said first cache contains the value as modified by another processing unit and that the modified value has not been written to said system memory device.

9. The computer system of claim **8** wherein said cache coherency means stores the modified value into said cache line of said second cache, prior to said issuing, and assigns a second coherency state to said cache line of said second cache indicating that said cache line contains the modified value.

10. The computer system of claim **9** wherein said cache coherency means further sets the hint bit to indicate that the second cache should deallocate the cache line, sources the modified value from said cache line of said second cache to a cache line of said first cache, and deallocates said cache line of said second cache.

11. The computer system of claim **10** wherein said cache coherency means further sends a snoop response from said second cache instructing said first cache to utilize the first coherency state.

12. The computer system of claim **10** wherein said cache coherency means further modifies the value in said cache line of the first cache in response to a store operation for an address associated with the value issued by said first processing unit, without issuing any bus transaction from said first cache.

13. The computer system of claim **12** wherein said cache coherency means further assigns the second coherency state to said cache line of said first cache in response to modifying the value in said cache line of said first cache.

14. The computer system of claim **10** wherein said cache coherency means further modifies a plurality of victimization priority bits associated with said cache line of said first cache in response to said assigning of the first coherency state to said cache line of said first cache.

* * * * *

	L #	Hits	Search Text	DBs
1	L1	83	(plural plurality multiple multiplicity several number) adj5 ((instruction adj2 (pointer counter) ip ((instruction address) near10 (fetch$3 prefetch$3)))) near20 (multithread$3 thread context)	USPAT; US-PGPUB

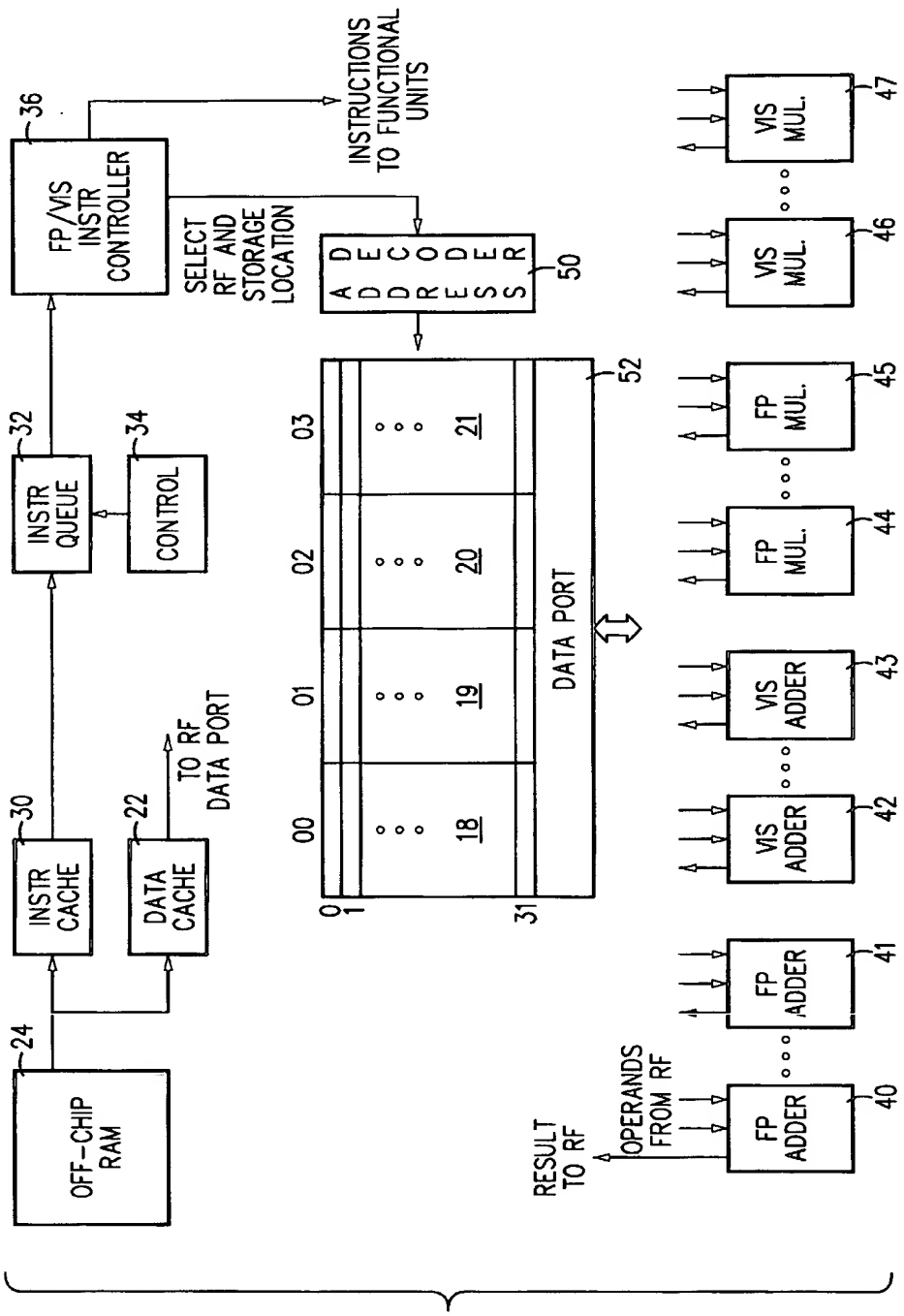

OFF-CHIP RAM
24
INSTR CACHE
30
DATA CACHE
22
TO RF DATA PORT
INSTR QUEUE
32
CONTROL
34
FP/VIS INSTR CONTROLLER
36
SELECT RF AND STORAGE LOCATION
INSTRUCTIONS TO FUNCTIONAL UNITS
ADDRESS
50
00
01
02
03
0
1
18
19
20
21
31
DATA PORT
52
RESULT TO RF
OPERANDS FROM RF
FP ADDER
40
FP ADDER
41
VIS ADDER
42
VIS ADDER
43
FP MUL.
44
FP MUL.
45
VIS MUL.
46
VIS MUL.
47

FIG. 6

	Document ID	U	Title	Current OR
1	US 20040 06871 7 A1	☐	System and method of utilizing a hardware component to execute an interpretive language	717/142
2	US 20040 06224 5 A1	☒	TCP/IP offload device	370/392
3	US 20040 05452 2 A1	☒	System and method to access web resources from wireless devices	704/8
4	US 20040 04960 0 A1	☒	Memory management offload for RDMA enabled network adapters	709/250
5	US 20040 04736 6 A1	☒	Method for dynamic flow mapping in a wireless network	370/466
6	US 20040 01989 1 A1	☒	Method and apparatus for optimizing performance in a multi-processing system	718/102
7	US 20040 01588 8 A1	☒	Processor system including dynamic translation facility, binary translation program that runs in computer having processor system implemented therein, and semiconductor device having processor system implemented therein	717/136
8	US 20040 01067 9 A1	☒	Reducing processor energy consumption by controlling processor resources	713/1
9	US 20030 21717 4 A1	☒	Establishing an IP session between a host using SIP and a device without an IP address	709/237
10	US 20030 15430 7 A1	☒	Method and apparatus for aggregate network address routes	709/245
11	US 20030 11279 4 A1	☒	System and method for multiple PDP contexts with a single PDP address at a GGSN	370/352
12	US 20030 09754 7 A1	☒	Context scheduling	712/228
13	US 20030 07184 7 A1	☒	Notification of messages to a terminal by means of vector image	345/772
14	US 20030 06021 0 A1	☒	System and method for providing real-time and non-real-time services over a communications system	455/452 .1
15	US 20030 02623 0 A1	☒	Proxy duplicate address detection for dynamic address allocation	370/338
16	US 20030 01447 3 A1	☒	Multi-thread executing method and parallel processing system	718/107
17	US 20030 00514 4 A1	☒	EFFICIENT CLASSIFICATION MANIPULATION AND CONTROL OF NETWORK TRANSMISSIONS BY ASSOCIATING NETWORK FLOWS WITH RULE BASED FUNCTIONS	709/235

g

Printed by HPS Server
for

Walk-Up_Printing

Printer: cpk2_2c21_gbkbptr
Date: 04/07/04
Time: 09:19:27

Document Listing

Document	Selected Pages	Page Range	Copies
US006374333	11	1 - 11	1
US006389512	14	1 - 14	1
US006405307	18	1 - 18	1
US006536034	16	1 - 16	1
Total (4)	59	-	-

	Document ID	U	Title	Current OR
18	US 20020191612 A1	☒	Method and apparatus for automatically determining an appropriate transmission method in a network	370/392
19	US 20020187788 A1	☒	Mobile communications system	455/450
20	US 20020181448 A1	☒	Prevention of spoofing in telecommunications systems	370/352
21	US 20020136206 A1	☒	Recursive query for communications network data	370/352
22	US 20020107964 A1	☒	Mobile communication system and data transferring method for use with mobile communication system	709/225
23	US 20020099844 A1	☒	Load balancing and dynamic control of multiple data streams in a network	709/232
24	US 20020062435 A1	☒	PRIORITIZED INSTRUCTION SCHEDULING FOR MULTI-STREAMING PROCESSORS	712/7
25	US 20020010710 A1	☒	Method for characterizing a complex system	715/500
26	US 20010037448 A1	☒	Input replicator for interrupts in a simultaneous and redundantly threaded processor	712/244
27	US 20010037447 A1	☒	Simultaneous and redundantly threaded processor branch outcome queue	712/239
28	US 20010036255 A1	☒	Methods and apparatus for providing speech recognition services to communication system users	379/88.01
29	US 20010029478 A1	☒	System and method for supporting online auctions	705/37
30	US 6721333 B1	☒	Point to point protocol multiplexing/demultiplexing method and apparatus	370/469
31	US 6707813 B1	☒	Method of call control to minimize delays in launching multimedia or voice calls in a packet-switched radio telecommunications network	370/356
32	US 6532554 B1	☒	Network event correlation system using formally specified models of protocol behavior	714/43
33	US 6519636 B2	☒	Efficient classification, manipulation, and control of network transmissions by associating network flows with rule based functions	709/223
34	US H002051 H	☒	System and method for providing multiple quality of service classes	370/395.21
35	US 6477562 B2	☒	Prioritized instruction scheduling for multi-streaming processors	718/108
36	US 6434676 B1	☒	FIFO with random re-read support and its application	711/154

US005845307A

United States Patent [19]

Prabhu et al.

[11] Patent Number: 5,845,307

[45] Date of Patent: Dec. 1, 1998

[54] **AUXILIARY REGISTER FILE ACCESSING TECHNIQUE**

[75] Inventors: **J. Arjun Prabhu**, Palo Alto; **Philip A. Ferolito**; **Eric T. Anderson**, both of Sunnyvale; **James A. Bauman**, San Jose, all of Calif.

[73] Assignee: **Sun Microsystems, Inc.**, Palo Alto, Calif.

[21] Appl. No.: **787,339**

[22] Filed: **Jan. 27, 1997**

[51] **Int. Cl.**[6] **G06F 12/02**

[52] **U.S. Cl.** **711/2**

[58] **Field of Search** ..... 364/DIG. 1 MS File, 364/DIG. 2 MS File; 395/376; 711/2, 5, 100, 152, 157, 170, 200, 215

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,070,703	1/1978	Negi	711/5
4,318,175	3/1982	Hawley	711/5
5,615,348	3/1997	Koino et al.	711/5

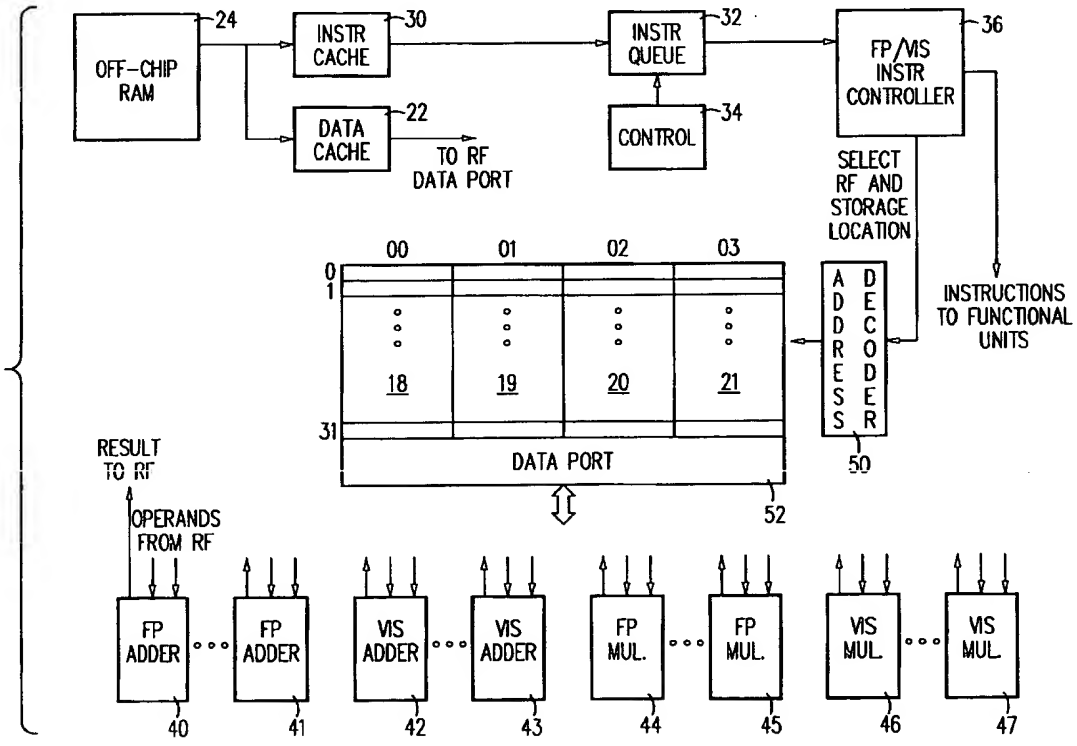
Primary Examiner—Robert B. Harrell

Attorney, Agent, or Firm—The Gunnison Law Firm

[57] **ABSTRACT**

Certain bits in existing op code formats for a processor do not change from one instruction to another when particular classes of instructions are used. Applicants optionally utilize one or more of these bits to identify one of a plurality of different register files from which to retrieve operands or to store the results of an operation. These bits along with allocated address bits in predetermined address fields now allow the processor to address many more registers. This can be used to increase the performance of the processor. Those programs not utilizing the bits outside of the address fields for designating a particular register file are backwards compatible with the modified processor.

16 Claims, 4 Drawing Sheets



	Document ID	U	Title	Current OR
37	US 6338078 B1	☒	System and method for sequencing packets for multiprocessor parallelization in a computer network system	718/102
38	US 6295600 B1	☒	Thread switch on blocked load or store using instruction thread field	712/228
39	US 6295557 B1	☒	Apparatus for simulating internet traffic	709/224
40	US 6292888 B1	☒	Register transfer unit for electronic processor	712/225
41	US 6272148 B1	☒	Scheme for reliable communications via radio and wire networks using transport layer connection	370/469
42	US 6233315 B1	☒	Methods and apparatus for increasing the utility and interoperability of peripheral devices in communications systems	379/88.01
43	US 6229880 B1	☒	Methods and apparatus for efficiently providing a communication system with speech recognition capabilities	379/88.01
44	US 6154777 A	☒	System for context-dependent name resolution	709/227
45	US 5854913 A	☐	Microprocessor with an architecture mode control capable of supporting extensions of two distinct instruction-set architectures	712/210

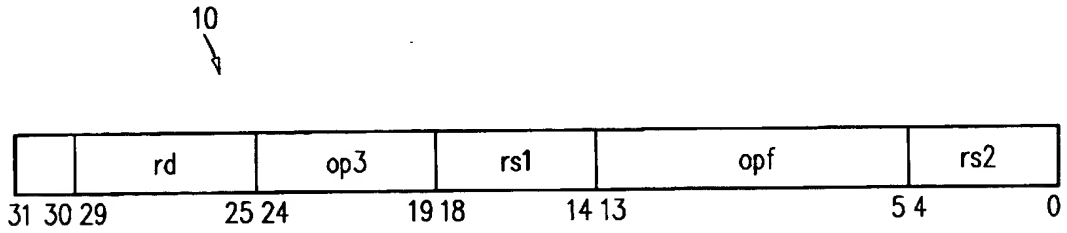


FIG. 1 (PRIOR ART)

Opcode	opf	Operation
FPADD16	0 0101 0000	FOUR 16-BIT ADD
FPADD16S	0 0101 0001	TWO 16-BIT ADD
FPADD32	0 0101 0010	TWO 32-BIT ADD
FPADD32S	0 0101 0011	ONE 32-BIT ADD
FPSUB16	0 0101 0100	FOUR 16-BIT SUBTRACT
FPSUB16S	0 0101 0101	TWO 16-BIT SUBTRACT
FPSUB32	0 0101 0110	TWO 32-BIT SUBTRACT
FPSUB32S	0 0101 0111	ONE 32-BIT SUBTRACT

FIG. 2 (PRIOR ART)

Opcode	op3	opf	Operation
FMULs	11 0100	0 0100 1001	MULTIPLY SINGLE
FMULd	11 0100	0 0100 1010	MULTIPLY DOUBLE
FMULq	11 0100	0 0100 1011	MULTPILY QUAD
FsMULd	11 0100	0 0110 1001	MULTIPLY SINGLE TO DOUBLE
FdMULq	11 0100	0 0110 1110	MULTIPLY DOUBLE TO QUAD
FDIVs	11 0100	0 0100 1101	DIVIDE SINGLE
FDIVd	11 0100	0 0100 1110	DIVIDE DOUBLE
FDIVq	11 0100	0 0100 1111	DIVIDE QUAD

FIG. 3A (PRIOR ART)

	Document ID	U	Title	Current OR
1	US 20040 06871 7 A1	☐	System and method of utilizing a hardware component to execute an interpretive language	717/142
2	US 20040 06224 5 A1	☒	TCP/IP offload device	370/392
3	US 20040 05452 2 A1	☒	System and method to access web resources from wireless devices	704/8
4	US 20040 04977 4 A1	☒	REMOTE DIRECT MEMORY ACCESS ENABLED NETWORK INTERFACE CONTROLLER SWITCHOVER AND SWITCHBACK SUPPORT	719/312
5	US 20040 04960 0 A1	☒	Memory management offload for RDMA enabled network adapters	709/250
6	US 20040 04736 6 A1	☒	Method for dynamic flow mapping in a wireless network	370/466
7	US 20040 03479 7 A1	☒	Domain-less service selection	713/201
8	US 20040 01989 1 A1	☒	Method and apparatus for optimizing performance in a multi-processing system	718/102
9	US 20040 01588 8 A1	☒	Processor system including dynamic translation facility, binary translation program that runs in computer having processor system implemented therein, and semiconductor device having processor system implemented therein	717/136
10	US 20040 01067 9 A1	☒	Reducing processor energy consumption by controlling processor resources	713/1
11	US 20030 21717 4 A1	☒	Establishing an IP session between a host using SIP and a device without an IP address	709/237
12	US 20030 21266 0 A1	☒	Database scattering system	707/1
13	US 20030 20448 2 A1	☒	Data search system	707/1
14	US 20030 19822 6 A1	☒	SEGMENTATION PROTOCOL THAT SUPPORTS COMPRESSED SEGMENTATION HEADERS	370/393
15	US 20030 15430 7 A1	☒	Method and apparatus for aggregate network address routes	709/245
16	US 20030 11279 4 A1	☒	System and method for multiple PDP contexts with a single PDP address at a GGSN	370/352
17	US 20030 10597 6 A1	☒	Flow-based detection of network intrusions	713/201

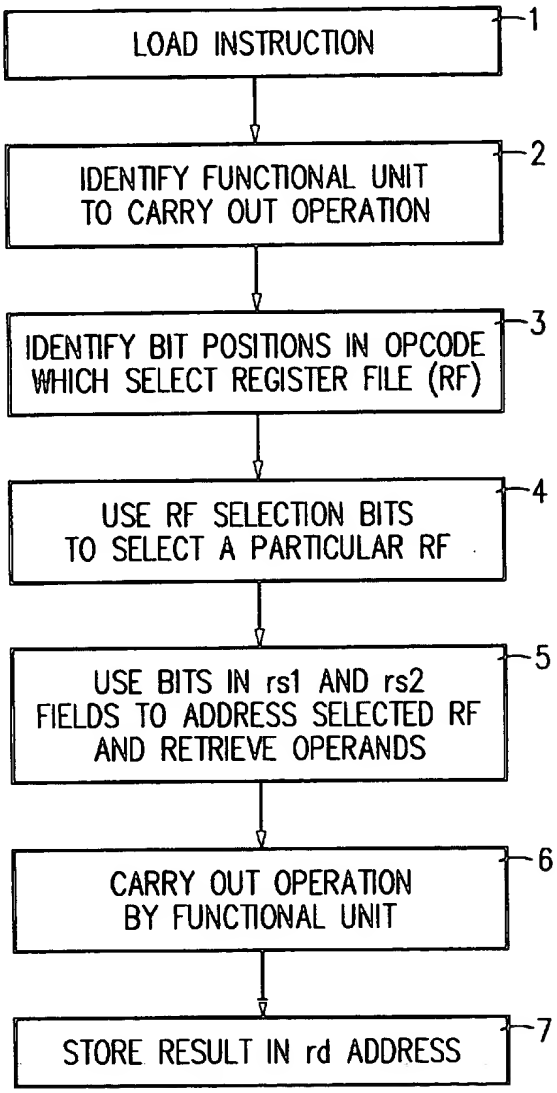

LOAD INSTRUCTION
1
IDENTIFY FUNCTIONAL UNIT TO CARRY OUT OPERATION
2
IDENTIFY BIT POSITIONS IN OPCODE WHICH SELECT REGISTER FILE (RF)
3
USE RF SELECTION BITS TO SELECT A PARTICULAR RF
4
USE BITS IN rs1 AND rs2 FIELDS TO ADDRESS SELECTED RF AND RETRIEVE OPERANDS
5
CARRY OUT OPERATION BY FUNCTIONAL UNIT
6
STORE RESULT IN rd ADDRESS
7

FIG. 7

	Document ID	U	Title	Current OR
18	US 20030097548 A1	☒	Context execution in pipelined computer processor	712/228
19	US 20030097547 A1	☒	Context scheduling	712/228
20	US 20030084337 A1	☒	Remotely controlled failsafe boot mechanism and manager for a network device	713/200
21	US 20030081582 A1	☒	Aggregating multiple wireless communication channels for high data rate transfers	370/338
22	US 20030071847 A1	☒	Notification of messages to a terminal by means of vector image	345/772
23	US 20030060210 A1	☒	System and method for providing real-time and non-real-time services over a communications system	455/452.1
24	US 20030057751 A1	☒	Dude tandem	297/243
25	US 20030033541 A1	☒	Method and apparatus for detecting improper intrusions from a network into information systems	713/201
26	US 20030026230 A1	☒	Proxy duplicate address detection for dynamic address allocation	370/338
27	US 20030014473 A1	☒	Multi-thread executing method and parallel processing system	718/107
28	US 20030014471 A1	☒	Multi-thread execution method and parallel processor system	718/107
29	US 20030005144 A1	☒	EFFICIENT CLASSIFICATION MANIPULATION AND CONTROL OF NETWORK TRANSMISSIONS BY ASSOCIATING NETWORK FLOWS WITH RULE BASED FUNCTIONS	709/235
30	US 20020191612 A1	☒	Method and apparatus for automatically determining an appropriate transmission method in a network	370/392
31	US 20020187788 A1	☒	Mobile communications system	455/450
32	US 20020181448 A1	☒	Prevention of spoofing in telecommunications systems	370/352
33	US 20020136206 A1	☒	Recursive query for communications network data	370/352
34	US 20020107964 A1	☒	Mobile communication system and data transferring method for use with mobile communication system	709/225

AUXILIARY REGISTER FILE ACCESSING TECHNIQUE

FIELD OF THE INVENTION

This invention relates to addressing techniques and, in particular, to a technique for addressing additional register files using a fixed instruction format.

BACKGROUND

The following background is in the context of a processor having a 32-bit architecture, but applies to virtually any architecture. FIG. **1** illustrates a 32-bit operation code (op code) **10** format for an UltraSPARC™ processor manufactured by Sun Microelectronics. The op code **10** contains various fields including an operation field (opf) identifying the specific operation to be performed, a first operand field (rs1) identifying the address of the first operand in a register file, a second operand field (rs2) identifying the address of the second operand in a register file, a result field (rd) identifying the address in a register file where the result of the operation is to be stored, and a general class field (op3) identifying the class of the operation to be performed such as a floating point (FP) operation, a visual instruction set (VIS) operation (typically used for graphics displays), an integer operation, or a load/store operation. Bit positions **30** and **31** identify a larger class. The bit positions of the various fields in the 32-bit op code **10** are identified in FIG. **1**.

Other architectures have similar predetermined locations for the various fields, and the present invention applies to a variety of architectures. Additional information regarding the UltraSPARC™ op code format and structure is found in the manual entitled Ultra SPARC™ User's Manual Revision 2.0—June 1996, by Sun Microelectronics, incorporated herein by reference.

It is very important that the various field lengths, op code organization, and instruction sets for a particular architecture not be changed, since the industry relies on that particular architecture and the existing instruction set for future applications.

The op code **10** identifies that a particular functional unit is to perform the operation in field opf on the operands located at the addresses in fields rs1 and rs2, then store the result in the address in field rd. In this particular architecture, five bits are allocated to each of the rs1, rs2, and rd fields. Hence, the register file can only have 32 addressable locations. In the UltraSPARC™ processor, each word in a particular address of the register file has 64 bits.

Functional units, such as a floating point adder and multiplier and a VIS functional unit, require the full 32 addressable locations in the register file for worst case processes.

It would be desirable to increase the number of functional units carrying out the operations in order to perform a number of operations simultaneously to increase the overall speed of the processor. However, the optimal performance benefit of adding more functional units is not realized if the number of addressable registers remains constant at **32**.

SUMMARY

Applicants have discovered a technique which can be used with a variety of architectures, such as the 32-bit architecture of the UltraSPARC™ processor, to access additional register files while using the existing op code format shown in FIG. **1**.

Applicants have identified certain bit positions in the op code format whose bits do not change from one instruction to another when particular classes of instructions are used. One example of such bits are in bit positions **12** and **13** of the op code format for the UltraSPARC™ processor for a VIS operation. In the instruction set for the VIS operations, bits **12** and **13** are always 00. Another example of such bits are in bit positions **9** and **13** for one class of floating point operations. Bits **9** and **13** are always 00.

In accordance with one embodiment of the invention, Applicants now optionally utilize bits **12** and **13** (or **9** and **13**) to identify one of up to four different register files, each having 32 addresses. These two bits along with the five bits in each of the rs1, rs2, and rd fields effectively allow these fields to address any one of 128 locations, rather than 32. This allows additional functional units to be employed in the processor to optimally increase the speed of the processor. Those desiring to not utilize bits **12** and **13** (or **9** and **13**) for designating a particular register file may simply set bits **12** and **13** (or **9** and **13**) to 00 as is presently done. Hence, the added flexibility is transparent if one does not chose to take advantage of it.

This concept may be applied to other architectures where the bits in certain bit positions are either always the same or unused.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** illustrates a 32-bit op code format for an UltraSPARC™ processor.

FIG. **2** illustrates a partial instruction set for a visual instruction set (VIS) operation, used for graphics displays, for an UltraSPARC™ processor.

FIG. **3A** illustrates a partial instruction set for a floating-point multiply or divide operation for an UltraSparc™ processor.

FIG. **3B** illustrates a more complete instruction set for the floating-point arithmetic operations revealing that bits **9** and **13** in the opf code are always set to 00.

FIG. **4** is similar to the op code format of FIG. **1** but modified to illustrate bit positions **12** and **13** in a VIS op code which identify a particular register file, out of a plurality of register files, which is addressed by the rs1, rs2, or rd fields.

FIG. **5** is the op code format of FIG. **1** modified to illustrate bit positions **9** and **13** in a floating point op code which identify a particular register file.

FIG. **6** illustrates the pertinent portions of a processor having a plurality of register files, where a particular register file is identified using bits in an op code format which were previously used for a different purpose.

FIG. **7** is a flowchart of a method carried out in accordance with one embodiment of the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

FIG. **2** illustrates the addition/subtraction subset of the visual instruction set (VIS) for a VIS functional unit, used typically for generating a graphics display created by a computer. FIG. **2** shows the alpha-numeric op code, the corresponding code in the opf field in the format of FIG. **1**, and a description of the operation performed. The identity of the operands on which the operation is performed and the storage location for the result of the operation are identified in the rs1, rs2, and rd fields in the op code format of FIG. **1**, previously described. The fact that the opf field identifies VIS instructions is conveyed in the op3 field (FIG. **1**) by the code 110110.

	Document ID	U	Title	Current OR
35	US 20020099844 A1	☒	Load balancing and dynamic control of multiple data streams in a network	709/232
36	US 20020087843 A1	☒	Method and apparatus for reducing components necessary for instruction pointer generation in a simultaneous multithreaded processor	712/228
37	US 20020087573 A1	☒	Automated prospector and targeted advertisement assembly and delivery system	707/102
38	US 20020062435 A1	☒	PRIORITIZED INSTRUCTION SCHEDULING FOR MULTI-STREAMING PROCESSORS	712/7
39	US 20020042821 A1	☒	System and method for monitoring and analyzing internet traffic	709/223
40	US 20020010710 A1	☒	Method for characterizing a complex system	715/500
41	US 20010056456 A1	☒	PRIORITY BASED SIMULTANEOUS MULTI-THREADING	718/103
42	US 20010048680 A1	☒	Method and apparatus for packet transmission with header compression	370/389
43	US 20010037448 A1	☒	Input replicator for interrupts in a simultaneous and redundantly threaded processor	712/244
44	US 20010037447 A1	☒	Simultaneous and redundantly threaded processor branch outcome queue	712/239
45	US 20010036255 A1	☒	Methods and apparatus for providing speech recognition services to communication system users	379/88.01
46	US 20010029478 A1	☒	System and method for supporting online auctions	705/37
47	US 20010010053 A1	☒	Service framework for a distributed object network system	718/105
48	US 6721806 B2	☒	Remote direct memory access enabled network interface controller switchover and switchback support	719/312
49	US 6721333 B1	☒	Point to point protocol multiplexing/demultiplexing method and apparatus	370/469
50	US 6707813 B1	☒	Method of call control to minimize delays in launching multimedia or voice calls in a packet-switched radio telecommunications network	370/356
51	US 6658447 B2	☒	Priority based simultaneous multi-threading	718/103
52	US 6625635 B1	☒	Deterministic and preemptive thread scheduling and its use in debugging multithreaded applications	718/102
53	US 6564267 B1	☒	Network adapter with large frame transfer emulation	709/250

FIG. 3A illustrates a partial instruction set for a floating-point multiply and divide functional unit which identifies the alpha-numeric op code, the code in the op3 field for identifying the functional unit, the code in the opf field for identifying the particular operation to be performed, and a description of the corresponding operation.

FIG. 3B is a more complete instruction set for the floating point operations, identifying the op code nomenclature as well as the corresponding 9-bit opf code (8:0) in bit positions 5–13 in FIG. 1.

Applicants have noted that the first two bits (bit positions 12 and 13) at the left of the opf code in the VIS of FIG. 2 are always 00 and that the first and fifth bits (bit positions 9 and 13) from the left of the opf code in the floating point instruction set are always 00. Hence, once it is identified that the instruction set is either VIS ($op3=36_{16}$) or a floating point ($op3=34_{16}$) operation, it is then known that the bits at bit positions 12 and 13 for a VIS operation and bit positions 9 and 13 for a floating point operation convey no information. Though a similar situation applies with other instruction sets either in the operation field or another field, only a VIS addition/subtraction instruction set and a floating-point instruction set are presented for illustration.

FIG. 4 illustrates an op code format 11 similar to that of FIG. 1 but modified to identify the two bit positions 12 and 13 in the operation field opf for a VIS operation, which now contains bits for identifying a particular register file among a plurality of register files.

FIG. 5 illustrates a similar op code format 16 for a floating point operation, which identifies the two bit positions 9 and 13 in the opf field used to identify a particular register file.

Since two bits in the VIS and floating point op code are available for designating a register file, due to these two bits being always the same, one of up to four register files may be identified in these two bit positions. Thus, since up to four register files, each having 32 addresses, can be identified by the two bits and each of the rs1, rs2, and rd fields may identify one out of 32 addresses, the registers have been increased from 32 to up to 128.

An application of this technique will now be described with respect to FIGS. 6 and 7. FIG. 6 illustrates modifications to a conventional processor for implementing the present invention. Since the architecture of processors is well known to those skilled in the art, a detailed description of the operation of processors is unnecessary to fully explain the present invention.

In FIG. 6, data is stored in designated addresses in a group of register files 18, 19, 20 and 21 via a data cache 22. An instruction in the form of the op code format 11 or 16 in FIGS. 4 or 5 is provided from an external RAM 24 into an instruction cache 30.

The instruction in the instruction cache 30 is then provided to an instruction queue 32 which in one embodiment has a capacity for 16 instructions. The next instruction in the queue 32, under the control of an instruction queue controller 34, is applied to an FP/VIS instruction controller 36, as identified in step 1 of FIG. 7. Controller 36 then dispatches the instruction to the proper functional unit 40–47 (step 2 in FIG. 7). Examples of functional units are illustrated as floating point and VIS adders 40–43 and floating point and VIS multipliers 44–47. Additional functional units would typically exist. Controller 36 determines the availability of the various functional units prior to tasking a particular functional unit to carry out the instruction. Controller 36 is typically a hardwired logic circuit.

Controller 36 addresses the register files to retrieve the operands for the designated functional unit as follows.

The locations of the one or more bits in the op code format 11 or 16 which identify the particular register file 18–21 to which the five address bits in the rs1, rs2, and rd fields apply depend upon the particular instruction set, as previously described. For example, for the VIS, the bits in bit positions 12 and 13 may be used to identify a particular register file since these two bit positions are essentially not utilized in the VIS (i.e., they are always 00). The controller 36 is programmed (in hardware or software) to associate particular bit positions, not necessarily adjacent, with the identification of a particular register file, shown in step 3 of FIG. 7, depending on the type of instruction (i.e., the op 3 field). The controller 36 will use the bit values in these bit positions to select a particular register file. If these particular bit positions occur in the opf field, the controller 36 will assign preprogrammed logical values (e.g., 00) to those bit positions when transmitting the opf instruction to the functional unit.

In step 4, the one or more bits in the bit positions containing the identification of the register file are decoded by decoder 50 to select one of the four register files 18–21. In the illustrated example, each of the register files 18–21 have 32 addresses and 64 bits per word. Register file 18 is identified by the 2-bit code 00 in bit positions 12 and 13 (assuming a VIS instruction), register file 19 is identified by the code 01, register file 20 is identified by the code 10, and register file 21 is identified by the code 11.

In step 5, the 5-bit operand address fields rs1 and rs2, illustrated in FIG. 4, are then used by controller 36 to address the two operands in the selected register file. Although the rs1 and rs2 fields are five bits, these fields can now address one of 128 locations using the present invention. These two operands are then retrieved and applied to the functional unit via a data port 52 of the register files. In the preferred embodiment, the FP and VIS adders 40–43 share the same data buses and the FP and VIS multipliers 44–47 share the same data buses.

In step 6, the functional unit carries out the operation on the operands.

In step 7, the functional unit stores the result of the operation in the address identified in the rd field, shown in FIG. 4, in the selected register file.

In certain operations, only a single operand is retrieved (read) from a register file and processed by a functional unit.

Thus, the number of addressable registers may be increased by four times without changing the op code format or the instruction set. If the programmer decides not to take advantage of the increased flexibility of the processor, the programmer would only use the traditional opf codes identified in FIGS. 2, 3A, and 3B to always access the first register file 18.

If a customer desires faster processing, the customer may purchase a processor containing additional functional units and a plurality of register files, such as shown in FIG. 6. In such a case, the programmer may then use the bits 12 and 13, for example, in the VIS op code to access any one of four register files, each having 32 addresses. The processor may be scaled to 32, 64, 96, or 128 registers, depending upon the needs of the customer, without affecting the instruction set or op code format for other customers not requiring the additional register files. This maintains backwards compatibility of the modified processor so that it can run on existing programs.

In the examples provided herein, the three 5-bit address fields in the op code effectively have been increased to 7 bits each by utilizing only two unused bits in the op code. This

	Document ID	U	Title	Current OR
54	US 6532554 B1	☒	Network event correlation system using formally specified models of protocol behavior	714/43
55	US 6519636 B2	☒	Efficient classification, manipulation, and control of network transmissions by associating network flows with rule based functions	709/223
56	US H002051 H	☒	System and method for providing multiple quality of service classes	370/395.21
57	US 6477562 B2	☒	Prioritized instruction scheduling for multi-streaming processors	718/108
58	US 6470443 B1	☒	Pipelined multi-thread processor selecting thread instruction in inter-stage buffer based on count information	712/205
59	US 6434676 B1	☒	FIFO with random re-read support and its application	711/154
60	US 6433795 B1	☒	System for integrating an on-line service community with a foreign service	345/738
61	US 6363424 B1	☒	Reuse of services between different domains using state machine mapping techniques	709/224
62	US 6338078 B1	☒	System and method for sequencing packets for multiprocessor parallelization in a computer network system	718/102
63	US 6295600 B1	☒	Thread switch on blocked load or store using instruction thread field	712/228
64	US 6295557 B1	☒	Apparatus for simulating internet traffic	709/224
65	US 6292888 B1	☒	Register transfer unit for electronic processor	712/225
66	US 6272148 B1	☒	Scheme for reliable communications via radio and wire networks using transport layer connection	370/469
67	US 6233315 B1	☒	Methods and apparatus for increasing the utility and interoperability of peripheral devices in communications systems	379/88.01
68	US 6229880 B1	☒	Methods and apparatus for efficiently providing a communication system with speech recognition capabilities	379/88.01
69	US 6223274 B1	☒	Power-and speed-efficient data storage/transfer architecture models and design methodologies for programmable or reusable multi-media processors	712/34
70	US 6216220 B1	☒	Multithreaded data processing method with long latency subinstructions	712/219
71	US 6209018 B1	☒	Service framework for a distributed object network system	718/105
72	US 6154777 A	☒	System for context-dependent name resolution	709/227
73	US 6105127 A	☒	Multithreaded processor for processing multiple instruction streams independently of each other by flexibly controlling throughput in each instruction stream	712/215
74	US 6073159 A	☒	Thread properties attribute vector based thread selection in multithreading processor	718/103
75	US 6052708 A	☒	Performance monitoring of thread switch events in a multithreaded processor	718/108
76	US 6020884 A	☒	System integrating an on-line service community with a foreign service	345/747

translates into using one-thirdas many bits as would be required had each of the address fields been expanded by two bits.

Those skilled in the art will understand that the concepts described herein may be applied to any other architecture, such as 16-bit or 64-bit architecture, and any op code format.

While particular embodiments of the present invention have been shown and described, it will be obvious to those skilled in the art that changes and modifications may be made without departing from this invention in its broader aspects and, therefore, the appended claims are to encompass within their scope all such changes and modifications as fall within the true spirit and scope of this invention.

What is claimed is:

1. A method carried out by a processor, said processor carrying out designated operations on at least one operand, after retrieving said at least one operand from one or more registers, and writing a result in a destination register, addresses of said at least one operand in said one or more registers and said destination register being identified by at least two N-bit fields in an operation code format, said method comprising:

identifying in said operation code format a first address field, having N bits, containing a first address of a first storage location within a register file, said register file having up to 2^N addressable locations;

identifying in said operation code format a second address field, having N bits, containing a second address of a second storage location within said register file;

identifying in said operation code format one or more register file selection bits outside of said first address field and said second address field for designating one of a plurality of register files as a selected register file for said first address and said second address, each register file in said plurality of register files having up to 2^N addressable locations; and

applying said first address and said second address to said selected register file to read said at least one operand from said selected register file and write said result into said selected register file.

2. The method of claim 1 wherein one or more bit positions storing said register file selection bits comprise bit positions 12 and 13 in an operation code format for a processor carrying out a visual instruction set operation.

3. The method of claim 1 wherein one or more bit positions storing said register file selection bits comprise bit position 12 in an operation code format for a processor carrying out a visual instruction set operation.

4. The method of claim 1 wherein one or more bit positions storing said register file selection bits comprise bit position 13 in an operation code format for a processor carrying out a visual instruction set operation.

5. The method of claim 1 wherein one or more bit positions storing said register file selection bits comprise bit positions 9 and 13 in an operation code format for a processor carrying out a floating point operation.

6. The method of claim 1 wherein one or more bit positions storing said register file selection bits comprise bit position 9 in an operation code format for a processor carrying out a floating point operation.

7. The method of claim 1 wherein one or more bit positions storing said register file selection bits comprise bit position 13 in an operation code format for a processor carrying out a floating point operation.

8. A method for programming a processor, said processor carrying out designated operations on at least one operand, after retrieving said at least one operand from one or more registers, and writing a result in a destination register, addresses of said at least one operand in said one or more registers and said destination register being identified by at least two N-bit fields in an operation code format, said method comprising:

providing in said operation code format a first address field, having N bits, containing a first address of a first storage location within a register file, said register file having up to 2^N addressable locations;

providing in said operation code format a second address field, having N bits, containing a second address of a second storage location within said register file;

providing in said operation code format one or more register file selection bits outside of said first address field and said second address field for designating one of a plurality of register files as a selected register file for said first address and said second address, each register file in said plurality of register files having up to 2^N addressable locations; and

providing an instruction in said operation code format for causing said processor to apply said first address and said second address to said selected register file to read said at least one operand from said selected register file and write said result into said selected register file.

9. The method of claim 8 wherein one or more bit positions storing said register file selection bits comprise bit positions 12 and 13 in an operation code format for a processor carrying out a visual instruction set operation.

10. The method of claim 8 wherein one or more bit positions storing said register file selection bits comprise bit position 12 in an operation code format for a processor carrying out a visual instruction set operation.

11. The method of claim 8 wherein one or more bit positions storing said register file selection bits comprise bit position 13 in an operation code format for a processor carrying out a visual instruction set operation.

12. The method of claim 8 wherein one or more bit positions storing said register file selection bits comprise bit positions 9 and 13 in an operation code format for a processor carrying out a floating point operation.

13. The method of claim 8 wherein one or more bit positions storing said register file selection bits comprise bit position 9 in an operation code format for a processor carrying out a floating point operation.

14. The method of claim 8 wherein one or more bit positions storing said register file selection bits comprise bit position 13 in an operation code format for a processor carrying out a floating point operation.

15. A processor comprising:

a plurality of functional units, each functional unit for carrying out a set of instructions on one or more operands and storing a result of an operation in a register;

a plurality of register files, each register file having up to 2^N addressable locations;

an input port receiving an instruction in an operation code format, said instruction including at least one operand address field containing a first address having N bits designating an address of a first storage location within

	Document ID	U	Title	Current OR
77	US 5900025 A	☒	Processor having a hierarchical control register file and methods for operating the same	712/248
78	US 5881277 A	☒	Pipelined microprocessor with branch misprediction cache circuits, systems and methods	712/239
79	US 5854913 A	☒	Microprocessor with an architecture mode control capable of supporting extensions of two distinct instruction-set architectures	712/210
80	US 5812811 A	☒	Executing speculative parallel instructions threads with forking and inter-thread communication	712/216
81	US 5796393 A	☒	System for intergrating an on-line service community with a foreign service	345/733
82	US 5724565 A	☒	Method and system for processing first and second sets of instructions by first and second types of processing systems	712/245
83	US 5357617 A	☐	Method and apparatus for substantially concurrent multiple instruction thread processing by a single pipeline processor	712/245

a register file, said instruction also including a designation address field containing a second address having N bits designating a second storage location for a result of an operation carried out by a designated functional unit, said instruction also containing one or more register file selection bits outside of said at least one operand address field and said destination address field for designating one of a plurality of register files as a selected register file for said first address and said second address, each register file in said plurality register files having up to 2^N addressable locations; and

a controller receiving said instruction and selecting one of said plurality of register files as a selected register file based on said one or more register file selection bits, said controller applying said first address to said selected register file to read an operand from said selected register file to apply said operand to a designated functional unit.

16. The processor of claim 15 wherein said designated functional unit writes said result into said second storage location in said selected register file after performing an operation on one or more operands from said selected register file.

* * * * *